CommCareHQ Deployment

Dimagi Inc

May 17, 2024

CONTENTS

1 Getting Started **3**
- 1.1 About CommCare HQ . . . 3
- 1.2 About this Guide . . . 3
- 1.3 A word of caution on Self Hosting . . . 3
- 1.4 How to use this guide to self host CommCare HQ . . . 4

2 Prerequisites to Setup CommCare HQ in Production **5**
- 2.1 Things to consider when Self Hosting . . . 5
- 2.2 Architecture and Platform Overview . . . 7
- 2.3 Roles And Responsibilities for Hosting CommCare . . . 7
- 2.4 Hardware requirements and Deployment Options . . . 10
- 2.5 Managing Hardware and the Physical Environment . . . 14
- 2.6 Software and Tools requirements . . . 16
- 2.7 CommCare Cloud Deployment Tool . . . 17

3 Deploy CommCare HQ **19**
- 3.1 Quick Install on Single Server . . . 19
- 3.2 Install Using Commcare-Cloud on one or more machines . . . 21
- 3.3 Troubleshooting first time setup . . . 32
- 3.4 Testing your new CommCare Environment . . . 36
- 3.5 Migrating CommCare HQ . . . 45
- 3.6 Go Live Checklist . . . 50

4 Operations and maintenance **53**
- 4.1 Managing The Deployment . . . 53
- 4.2 Monitoring and Alerting . . . 60
- 4.3 Set up Sentry for error logs . . . 77
- 4.4 Expectations for Ongoing Maintenance . . . 78
- 4.5 Supporting Your Users . . . 78

5 How to Scale **87**
- 5.1 Performance Benchmarking for CommCare HQ using Locust . . . 87
- 5.2 How to Estimate Infrastructure Sizing . . . 89

6 CommCare HQ Services Guides **91**
- 6.1 PostgreSQL . . . 91
- 6.2 BlobDB . . . 109
- 6.3 Nginx . . . 113
- 6.4 Kafka . . . 116
- 6.5 Pillowtop . . . 120
- 6.6 RabbitMQ . . . 122

6.7 Redis . . . 122
6.8 Set up Bitly for generating app codes . . . 123
6.9 Keepalived . . . 123

7 Backups and Disaster Recovery **125**
7.1 Backup and Restore . . . 125
7.2 ElasticSearch Backup on Swift API . . . 131
7.3 Disaster Recovery . . . 132

8 Securing CommCare HQ **135**
8.1 Introduction . . . 135
8.2 Application Security . . . 136
8.3 Host and Disk Security . . . 136
8.4 Network and Physical Security . . . 136

9 Reference Annexure **137**
9.1 CommCare Cloud Reference . . . 137
9.2 User Access Management . . . 187
9.3 Firefighting Production Issues . . . 188
9.4 Specialized Howtos . . . 222
9.5 Settings in `public.yml` . . . 233
9.6 Ports Required for CommCare HQ . . . 235

10 About this changelog **237**
10.1 Changelog . . . 237

This documentation contains guides and reference material on how to self host CommCare HQ locally. It covers requirements to self host CommCare HQ, various deployment types, infrastrucutre and team requirements to host CommCare HQ, installation instructions for single and multi server deployment and how to manage CommCare HQ deployment through the entire hosting lifecycle.

CHAPTER

ONE

GETTING STARTED

1.1 About CommCare HQ

CommCare is a multi-tier mobile, server, and messaging based platform. The server part of the platform is called as CommCare HQ. The platform enables end-users to build and configure content through a user interface, deploy that application to Android or J2ME based devices, and receive data back in real-time (if connected) from the mobile applications. In addition, content may be defined that leverages bi-directional messaging to end-users via API interfaces to SMS Gateways, E-mails systems, or other messaging services. The system leverages multiple persistence mechanisms, analytical frameworks, and open source libraries.

CommCare HQ is offered through Dimagi managed cloud with free and tiered subscription models at https://wwww.commcarehq.org. Please refer to https://www.commcarehq.org/pricing/ and https://dimagi.com/services/ for information on this. If you are interested in self hosting CommCare HQ this guide is for you.

1.2 About this Guide

This documentation contains tutorials, guides, and reference material on how to self host CommCare HQ, infrastructure and team requirements, installation instructions for single and multi server deployment, and how to manage CommCare HQ deployment through the entire hosting life-cycle. This documentation is relevant for those who want to host CommCare HQ locally on their infrastructure.

If you are looking for something else please see below.

- If you are interested in using CommCare HQ without having to host yourself, please check out our cloud offering at https://www.commcarehq.org.
- Docs on how to use CommCare HQ https://wiki.commcarehq.org/display/commcarepublic/Home
- Setting up CommCare HQ locally for development https://github.com/dimagi/commcare-hq/blob/master/DEV_SETUP.md

1.3 A word of caution on Self Hosting

CommCare HQ is a complex, distributed software application, made up of dozens of processes and several pieces of third-party open source database software. It has been built for scale rather than simplicity, and as a result even for small deployments a CommCare server or server cluster can be challenging to maintain. An organization endeavoring to manage their own CommCare server environment must be willing to devote considerable effort and system administration capacity not only in the initial phases of provisioning, setup, and installation, but in steady state as well.

If you or your organization is hosting or interested in hosting its own CommCare HQ server environment, we strongly suggest you to read about the *Things to consider when Self Hosting* and prerequisites required to host CommCare locally at *Prerequisites to Setup CommCare HQ in Production*.

1.4 How to use this guide to self host CommCare HQ

1. Read and understand all the prerequisites for hosting at *Prerequisites to Setup CommCare HQ in Production*.
2. Figure out which installation is most suitable for you using *Hardware requirements and Deployment Options* doc.
3. Use one of the deployment guides *here* to install and configure all the required services to run CommCare HQ.
4. For managing, updating CommCare HQ, monitoring and troubleshooting check out *Operations and maintenance*.
5. To understand how each service is used and perform common operations related to services checkout the section on *CommCare HQ Services Guides*.
6. To scale services when required consult the section on *How to Scale*.
7. Ensure you have setup backups correctly and have a disaster recovery plan using *Backups and Disaster Recovery*.

CHAPTER

TWO

PREREQUISITES TO SETUP COMMCARE HQ IN PRODUCTION

In this section, we list down various prerequisites for self hosting CommCare HQ successfully.

- To know the architecture of CommCare HQ and complexities of hosting read *Architecture and Platform Overview* and *Things to consider when Self Hosting*.
- To know the personnel and technical skills required read *Roles And Responsibilities for Hosting CommCare*.
- To know how much hardware you need and what tools and services you need read *Hardware requirements and Deployment Options* and *Software and Tools requirements*.
- Once you have all the prerequisites you can follow tutorials in *Deploy CommCare HQ* section to install CommCare HQ.

2.1 Things to consider when Self Hosting

If you are interested in running a CommCare server environment there are a number of serious obstacles to consider before taking the plunge. If you are already managing a CommCare server environment and are finding it challenging, you may also find thinking about the considerations listed here helpful before deciding your next steps.

2.1.1 CommCare Cluster Management

CommCare HQ is a complex, distributed software application, made up of dozens of processes and several pieces of third-party open source database software. It has been built for scale rather than simplicity, and as a result even for small deployments a CommCare server or server cluster can be challenging to maintain.

Many processes

While the setup automation should work out of the box, once you're up and running you will have to troubleshoot any issues yourself. This means being familiar enough with every piece of third-party database software to be able to debug issues and bring it back to life or recover from some other bad state; it also means understanding what each of the many types of CommCare application processes does well enough to troubleshoot issues.

CommCare HQ relies on the following open source technologies:

- PostgreSQL
- PL/Proxy and a custom sharding setup
- `pg_backup` and streaming replication
- CouchDB
- Redis

- Riak/Riak CS (mandatory for multi-server environments)
- Elasticsearch
- Kafka (and Zookeeper)
- RabbitMQ
- Nginx

CommCare HQ also employs different types of application processes each with its own distinct purpose and set of functionality:

- Web workers
- Asynchronous task processors (Celery)
- ETL processes ("Pillowtop") that themselves come in a number of different flavors and fill a number of different roles
- The CommCare mobile engine exposed as a webservice ("Formplayer")

You will also need some familiarity with the following sysadmin tools:

- Monit
- Supervisor
- Ansible
- EcryptFS
- LVM (optional)

2.1.2 Physical Server Management

This section is for people interested in owning the physical hardware running their servers. If you will be using an established data center or modern hosting provider, this doesn't apply.

It can be tempting to want to run CommCare HQ on server hardware that you own. After all, a server is just a computer connected to the internet, with enough RAM, CPU, and Disk to run the software it's hosting. However, in practice, there's a lot more that goes into it.

Network Connection

One of the biggest difficulties in maintaining physical server hardware is that to be reliable it must **always** be connected to the internet over a stable and reliable network. In many of the areas of the world that are most interested in self-hosting for data-sovereignty reasons, network connectivity is notoriously spotty. Real data centers will go to great lengths to keep their servers always connected to the public internet, including network redundancy. It can be close to impossible to do this at the level of one or two servers in an office, so in practice, these servers will experience large rates of downtime.

Power Source

Another often-overlooked challenge with managing physical hardware is maintaining a reliable power source. In many areas of the world that use CommCare, reliable power is not a given. Even with a generator, unless there's a sophisticated battery backup system in place, it will take a number of seconds for the generator to kick in, and even the slightest blip in power supply will cause a server to shut off without warning. Thus, without a well-planned system in place for maintaining consistent power even through a grid outage, these servers will experience a lot of downtime and will be at risk for data corruption as a result of powering off without warning.

Physical Security

Real data centers are highly guarded by security personnel and have authorization protocols preventing physical access to the servers except by a small handful of entrusted individuals. With physical access to a machine, a malicious or unwitting person can steal private information or cause untold damage including deletion of all data. In the typical office environment, maintaining an appropriate level of physical security around the hardware is not a given, and will require concerted effort and planning.

Temperature Control

Computer hardware gives off a tremendous amount of heat. Servers are **hot**, and so are many of the areas of the world that use CommCare. Real data centers are carefully designed to manage heat flow and sites are often specifically chosen for environmental features such as cold outdoor temperature or proximity to a vast water source for cooling. Overheating is a very real issue and when it happens it will lead to unpredictable hardware failure.

2.2 Architecture and Platform Overview

It is useful to have some context on what it is that you are hosting when you are hosting CommCare HQ. A good place to find that context is the Architecture Overview. This will also show how data flows for forms and cases.

For a more abstract overview of the platform, visit the Platform overview link.

2.3 Roles And Responsibilities for Hosting CommCare

2.3.1 Introduction

This section is intended to convey the different roles and responsibilities which are associated with setting up and maintaining a privately hosted local cloud instance of CommCare HQ, along with expectations around what skills and knowledge are needed to fulfill them. An individual person or firm may fulfill a number of these roles, but they are separated here to clarify the skills and availability required to fulfill them.

These are based directly on Dimagi's process and experience providing CommCare HQ as a managed SaaS platform and managed cloud, but they may not be comprehensive depending on specific local or organizational needs.

2.3.2 Infrastructure Provider

Responsible for providing the physical resources needed by the cluster (servers, network interfaces, etc.) When providing infrastructure as a service to other businesses we'd refer to this as the "Cloud Services Provider."

Scope of Work

- Virtual Machine hosting
- Support and debugging for non-responsive hardware

Skills and Knowledge Required

- VMWare or other virtualization technology
- Network interfaces, including firewalls and load balancing

Sample Providers

- AWS EC2
- Rackspace Managed Infrastructure
- Microsoft Azure
- IBM Softlayer

2.3.3 Cluster Management

Responsible for understanding and maintaining the physical infrastructure that has been provided and ensuring its availability and reliability.

Scope of Work

- Installing operating systems and configuring cloud resources to spec
- Applying security patches and operating system updates
- Establishing protocols for backups and system recovery
- Monitoring the servers for intrusion detection and other active security monitoring
- Responding to active threats to availability and security like DDOS attacks

Skills and Knowledge Required

- General Linux system administration
- Understanding of virtual machine resources, network infrastructures, and other core IT concepts
- Familiarity with backup and recovery schemes for the relevant components of the CommCare infrastructure
- General knowledge of security best practices

Sample Providers

- Rackspace Managed Services
- AWS Managed Services

Note: Commercially, this offering is generally made on top of managed infrastructure by the same company, and the two are traditionally bundled.

2.3.4 Cloud Application Management / Operations Engineer

Responsible for deploying the CommCare web application and dependent services, and keeping them up-to-date.

When this role and the role above are provided together as a service, that shared role is referred to as a "Managed Services Provider".

Scope of Work

- Sizing and scoping the cloud infrastructure needs
- Provisioning servers with the cloud software applications
- Debugging and maintaining individual services that are part of the cloud
- Keeping the cloud software up to date with new changes
- Supporting data migrations and other 'on server' (as opposed to 'in application') operations tasks
- Monitoring the status and availability of individual services and addressing issues

Skills and Technologies

- Familiarity with Python and ability to interpret Python tracebacks
- Familiarity with Linux system administration and management
- Experience with deploying and maintaining a cluster of servers hosting a web application which is dependent on federated services

Expected to learn in the first 1-3 months of working on CommCare Cloud. *Please note that the deployment of the local system won't be successful unless the team as a whole has experience or some level of comfort with these tools*:

- Familiarity or capacity to learn the core components of a CommCare cloud + Web Proxy Server (nginx) + SQL database (PostgreSQL) + Messaging queue (RabbitMQ) + Cache server (Redis) + Search index (Elasticsearch) + Object Storage (RiakCS) + Distributed Message Log (Kafka) + Distributed Configuration management (Zookeeper)
- Familiarity with the frameworks relied upon by our operations tools + Ansible + Monit + Supervisor + EncryptFS + LVM

Sample Providers

"Managed Application Hosting":

- Rackspace SAP Hosting
- IBM Managed Application Services

2.3.5 CommCare Application Administrator

Responsible for configuring CommCare HQ from inside of the web application.

Scope of Work

- User and application configuration
- Processing tech support direction when internal maintenance tools need to be run within the HQ web app
- Providing technical support for end users of the application

Skills and Technologies

- Familiarity with CommCare HQ

2.4 Hardware requirements and Deployment Options

2.4.1 Introduction

CommCare HQ is a complex software platform, composed of several parts (application builder, report builder, databases, etc.). The architecture of CommCare HQ allows hosting for wide variety of usage loads. In this guide we list down various server configurations suitable for various user loads to start with.

When determining what configuration to use, it is important to keep the expected growth of the project in mind, in order to optimize costs and limit possible service interruptions due to server management operations. For example, if there is a delay of days, weeks or months between the time when more resources are requested, and the time when they becoming available, then it is better to have some buffer. Make sure there is more drive space, RAM, or cores than you need, so that by the time the resources you are going to need have arrived, it is not too late.

Alternatively, if you are hosting on a platform where requisitioning resources is fast, or instant, buffer is less important, and it would make more sense to optimise on resource costs.

Importantly, optimum resource management requires accurate resource monitoring. For more details on this topic, see the CommCare Cloud documentation on label_datadog-for-monitoring.

2.4.2 Recommended Server Sizings for different loads

The following table summarizes common server configurations. This should provide a basic understanding of the implications of scaling a project to several thousands of users.

Server Configuration	Description	Scalability (users)	Infrastructure Cost	Personnel Cost	Risks
Single Server	A single server on which all the pieces of the CommCare HQ software suite are installed.	< 1500	Low	Mediun	No built in redundancy. Difficult to transition to larger cluster
Micro Cluster	Two servers running in parallel which provides higher capacity than a single server as well as better redundancy characteristics.	< 3500	Low	Mediun	
Small Cluster	A five-server cluster, which the parts of the CommCare HQ software suite are distributed across.	< 15K	Medium	High	Complex setup requiring more advanced management
Large Cluster	A cluster of more than five servers, which the parts of the CommCare HQ software suite are distributed across.	> 15K	High	Very high	Very complex setup and management

The number of users given in this table is a very rough guide. Every project is different: Some have many users, each submitting a small number of forms. Some have few users submitting many forms. Some CommCare apps are simple, and servers do not need to spend a lot of resources on syncing data and processing form submissions. Other apps are complex, and the same number of users demand much more from the servers. Some apps are used via the web interface exclusively, or have a lot more multimedia, or submit large form attachments, shifting requirements from some parts of CommCare HQ to other parts.

The recommended starting point for a project which is starting with a small number of users but plans to scale beyond 1500 users, is the Micro Cluster configuration. It gives the best combination of scalability and cost.

By monitoring which services are using what resources, that data will determine how to grow the cluster. For practical guidance on this topic, see the CommCare Cloud documentation on label_datadog-for-monitoring.

Virtualization

Where possible, virtual servers hosted by a Cloud Service Provider (CSP) should always be preferred over physical servers. The reason for this is that it makes maintenance and scaling of the cluster much simpler. It is also generally more cost effective as the utilization of the resources can be optimized and the burden of hardware management is removed.

It is recommended that virtualization be seriously considered for any cluster that would require more than three physical servers.

Adding virtualization to physical servers makes it possible to utilize the physical resources better, and makes certain maintenance tasks simpler. However, installing and managing a virtualization layer requires highly skilled personnel.

2.4.3 Single Server

This configuration is appropriate for projects with fewer than 1500 users, each user with an active case load of fewer that 10,000 cases, and the project has no intention of growing.

Specific hardware guidelines can become outdated quickly. The following specifications might be a reasonable suggestion:

- A mid-range server CPU
- At least 32 GB memory
- 2 TB SSD storage

Amazon EC2 t2.2xlarge and t3.2xlarge instances, configured with sufficient General Purpose EBS storage, meet these CPU, memory and storage specifications.

It is important to note that these suggestions are not appropriate for all projects. Monitoring resource usage, and adapting server resources accordingly, is crucial to meeting the needs of any project.

A Single Server configuration does not offer high availability. Server failures can take the environment offline. If uptime is crucial for the project, this is not a good option; a Micro Cluster configuration would be more appropriate.

2.4.4 Micro Cluster

From a resource perspective, this configuration is two single servers. But it opens up possibilities for how the environment is configured: One server can be configured as fail-over for the services on the other server. This is important for projects that need high availability. Alternatively, services can be balanced across both servers. If the project needs the resources of both servers combined, this sacrifices high availability, but passes the initial hurdle to building a larger cluster as a project grows.

This configuration is appropriate for small projects (projects with fewer than 1500 users, each user with an active case load of fewer that 10,000 cases) that need high availability.

It is also appropriate as a starting configuration for small projects that intend to grow to medium-sized projects, because it is more difficult to turn a Single Server configuration into a cluster than it is to extend a Micro Cluster configuration.

And it is appropriate for projects with fewer than about 3500 users.

Depending on the size of the project, this configuration has more range in terms of resource specification. For a small project, without high availability, resources for each machine could be lower than for a Single Server configuration:

- A mid-range server CPU
- At least 16 GB memory
- 1 TB SSD storage

Amazon EC2 t2.xlarge and t3.xlarge instances, configured with sufficient General Purpose EBS storage, meet these specifications.

For a small project which needs high availability, or for a medium-sized project, twice the requirements of the Single Server configuration would be appropriate:

- A mid-range server CPU
- At least 32 GB memory
- 2 TB SSD storage

Amazon EC2 t2.2xlarge and t3.2xlarge instances, configured with sufficient General Purpose EBS storage, meet these specifications.

2.4.5 Small Cluster

A five-server cluster may be appropriate for projects with up to about 15,000 users. By this point virtualization should be considered mandatory, for the sake of scalability, and in order to optimize hardware resource usage.

If the size of the project allows, start with virtual machine instances that are not at the highest resource specification. This allows for some buffer to scale vertically (in other words, add more resources to the same virtual machine) before the necessity to scale horizontally (add more virtual machines).

Amazon EC2 t2.xlarge and t3.xlarge instances meet this description.

Storage requirements will be determined by the function of each server; proxy and web servers will require less storage, database servers will require more.

The level of skills, and the number of personnel, required to manage a Small Cluster configuration are higher than for a Single Server or a Micro Cluster.

2.4.6 Large Cluster

Depending on the nature of a project, typically as it approaches or surpasses 15,000 users, it will require a server cluster of more than five servers.

Recommendations are the same as for a Small Cluster configuration:

- Allow some room to scale virtual machines vertically before needing to scale horizontally
- Monitoring is crucial, because decisions must be guided by data

The level of skills, and the number of personnel, required to manage a Large Cluster configuration are higher than for a Small Cluster.

2.4.7 Running CommCare HQ inside Docker

Running CommCare HQ inside docker is currently not supported/optimized for production environments. However, it is the recommended way to get the CommCare HQ ecosystem up and running for development purposes.

Running services inside containers requires additional resources to the base resource requirements for the ecosystem running inside the container(s). The minimum resource requirements to run the CommCare HQ ecosystem with docker is the following:

- CPU: 31.133% (on a single core)
- Memory: 5GB

However, it is recommended that your system have more resources than the above in order to do more than just “keep the lights on”. The exact resource requirement is heavily dependend on your own system and you should use the above only as a baseline.

2.5 Managing Hardware and the Physical Environment

Dimagi recommends outsourcing the maintenance of hardware and the physical hosting environment to a cloud service provider. For some projects, for a range of possible reasons or on balance of factors, that may not be desirable, or an option.

2.5.1 Third-party documents

Perhaps the canonical document on managing hardware, within the wholistic context of managing a data center, is the BICSI International Standard, ANSI/BICSI 002, Data Center Design and Implementation Best Practices. At the time of writing, the latest revision is BICSI 002-2019. At 500 pages long, it is comprehensive. A digital copy costs $525.

Samples, and a presentation based on the content are available free online. The presentation will give a good impression of the detailed thinking required to maintain a secure and reliable hosting environment.

2.5.2 Costs for maintaining a self-hosted production environment

These are long-term (year-over-year) costs that should be considered when determining the price of a "fixed cost" option, where servers are procured specifically for the project.

Purchasing production server equipment is quite different from purchasing personal computing equipment. Since servers will run continuously, and operations need to be available within predictable prices, server hardware cost is built around warranty and license life-cycles for equipment, where the warranty will ensure that the hardware continues to deliver over the time period.

For this example, we will presume a representative 16-Core 64GB server with a base cost of about $5,000, which generally fulfills the requirements of a monolith server. We will presume that the project will run for a five year period for representative total costs.

Hardware Replacement and Warranty

The expected lifespan for reliable server equipment in a data center is 3-5 years, which depends on many factors but can be lowered depending on the quality of power supplied and things like appropriate cooling and ventilation.

Traditionally a team without a significant stock of replacement components will purchase servers with a comprehensive warranty plan with the expectation that they will purchase new equipment when the warranty expires.

"Next business day" (v. 24/7 which is more expensive) support for 3 and 5 years costs $900 and $3000 respectively, making the year to year cost for the server:

Years in Service	Base	Warranty / Support	Yearly Cost
3	$5000	$900	$1,966
5	$5000	$3000	$1,600

Storage

When made available from a vendor, high durability block storage is made available at a fixed price. This storage will be robust against failures in individual hardware disks transparently without intervention. Without high durability storage, it should be expected that disk failure will result in the need to restore the system in full from backup (and with the loss of data in between the backup) at least once in a three year period.

When block storage isn't available, high durability storage must be created and maintained manually. This requires available backup disks, a software or hardware SAN appliance to produce high durability virtual storage, and a maintenance plan for replacing disks urgently.

Hard drive warranties are generally around 3 years, and producing 1TB of storage would generally require purchasing 1.5TB worth of disks during that period, since 33% will be lost to the redundant storage layer, and 1 additional disk needs to remain available to replace a faulted disk.

A 500GB server disk from HP is ~$300, making the year-to-year cost for 1TB of storage (including replacement) around $300 per year (3 years per disk, 3 disks in use at once for 1TB of storage).

Disk Lifetime	Cost Per Disk	Disks per TB	Yearly Cost per TB	Cost per year[1]
3	$300	3	$300	$1500

Networking

It is critical for the security of the server that they be on an isolated network segment which is provided by hardware with up-to-date firmware. Since network traffic will be directed inward to the data center from the external WAN connection, out-of-date unsupported firmware is a critical security vulnerability.

An example would be a Dell Sonicwall Next Generation Firewalls (NGFW) gateway appliance, like a TZ300. These networking appliances have an initial purchase cost and an ongoing licensing cost for keeping firmware up to date and in-spec. Example costs are described below

Appliance	Initial Purchase Cost	Yearly License Cost
SonicWall TZ300	$625	$400

Monitoring

The IT administrator managing the system will require the ability to monitor the systems. For example, if hard drives are configured in a RAID-10 configuration and a drive fails it is critical for an administrator to be notified immediately to ensure the drive is replaced rapidly and the system is not at risk of data loss.

A standard on-premise tool for managing servers is Nagios, with cost details provided below

Appliance	Initial Purchase Cost	Yearly License Cost
Nagios	$1,995	$1,695

[1] The recommendation for a monolith is 1TB of storage per year of project. On flexible block storage, this cost could be calculated with storage as needed (i.e. 1TB year 1, 2TB year 2, 3TB year 3, etc.), but with self-hosted RAID storage extending the storage volume requires new disks, which is complex. A minimum cost would be 3TB for years 1, 2, and 3, then 5TB years 4 and 5, which would require a complex migration requiring subject matter expertise.

Backups / Disaster Recovery

In addition to the primary production environment, there needs to be a location for backups to be stored, which will require a second redundant system which may or may not already be available at the existing datacenter.

Prices for storing backups can vary dramatically based on the backup architecture, but the most basic back-of-envelope costs would be to double the cost of storage (possibly with slower disks), if an existing system is available to host backups. Alternatively a second, lower power host server can be procured as well.

Support Licenses

Cloud hosting data centers which are managing servers at the OS level will generally have licensed support for Operating Systems. This support ensures that IT Admins are able to provide seamless uptime for servers and resolve critical issues at the OS level if they occur.

Representative Yearly Costs

Taking into account the above, a reasonable estimate for a baseline yearly cost over a 5 year period would include the following

Hardware	Storage	Networking	Monitoring	Yearly	Monthly
$1,600	$1,500	$400	$1,695	$5,195	$433

Some of these costs may be shared between different server systems, reducing the monthly cost, which is how Cloud providers are able to make these systems available more cheaply.

Due to the more complex pricing these costs do not include

- "One-time" costs for extra disks or initial purchases of Nagios/network appliances
- Backups
- Costs of maintenance
- Subject Matter Expert (SME) support for OS's or services
- Costs to replicate Information Security Management (ISMS) protocols including Physical Access control and regular auditing

All of which are important considerations for individual programs

2.6 Software and Tools requirements

A production grade CommCare HQ instance with all the features available requires good number of third-party software. Below is a list of softwares and tools required to run an instance.

1. Ubuntu 22.04 Server as operating system on all environments.
2. *CommCare Cloud Deployment Tool* to deploy all the other services.
3. Email and SMS gateways for features related to Email and SMS to work.
4. sentry.io for error logging. If you would like to self host Sentry using commcare-cloud, see sentry-on-prem.
5. Datadog or Prometheus for monitoring and alerting.
6. Google Maps or Mapbox API keys if you are using features that use these APIs.

2.7 CommCare Cloud Deployment Tool

2.7.1 What is commcare-cloud?

`commcare-cloud` is a python-based command line tool that uses the open source technology Ansible to automate everything you need in order to run a production CommCare cluster.

While it is possible to install on a laptop with a linux-like command line interface, it is primarily designed to be run on the machine that is hosting CommCare. (If you are hosting CommCare on more than one machine, `commcare-cloud` only needs to be installed on one of them.) In this documentation, we will call the machine on which `commcare-cloud` is installed the "control machine". If you are hosting CommCare on a single machine, that machine is also the control machine.

For installation see *Installation* For list of available commands see cchq-commands

2.7.2 What is the Control Machine

The machine where `commcare-cloud` is installed is known as the control machine. It is a single machine where you will be able to run any service checks, deploy code changes and update CommCare HQ code.

If you are running a monolith installation per *Deploy CommCare HQ* this will be the same machine that you installed all the CommCare HQ services on.

We recommend that the control machine be in the same datacenter or network as the rest of your server fleet.

2.7.3 Setting up a control machine

1. Install `commcare-cloud` *Installation* on it.
2. Configure `commcare-cloud` with *inventory.ini* of your server fleet.
3. Update the known-hosts file to access all the servers by running .. code-block:: bash

       $ commcare-cloud <env> update-local-known-hosts

2.7.4 User Management

User access to all machines on the fleet is managed through the control machine. User permissions are stored in the `_users` and `_authorized_keys` directories in the environment.

See more about these files and how to update them in the *_users* section in environment documentation.

2.7.5 Accessing the control machine

Once users are correctly added, they should access the control machine with key-forwarding enabled from their own computers. From a Unix machine:

```
$ ssh username@{control_machine IP} -A
```

If you are a Windows user using PuTTY to access the control machine, follow the instructions on this SuperUser answer to enable key forwarding.

This will allow those users to subsequently ssh into any of the other machines in the fleet, and run any `commcare-cloud` commands directly from the control machine.

2.7.6 commcare-cloud reference

Check out *CommCare Cloud Reference* for more information on `commcare-cloud`.

CHAPTER

THREE

DEPLOY COMMCARE HQ

This section has details on the following topics.

- How to deploy a CommCare HQ instance on one or more servers (also referred as install sometimes).
- Import data if you are migrating from another existing instance such as Dimagi's www.commcarehq.org or any other using *Migrate a Project from one instance to a new instance*.
- A *Go Live Checklist* and *basic QA tests* to make sure everything is working well before making your instance live to the public.

Once you have understood what deployment option is most suitable for you using *Hardware requirements and Deployment Options* guide and have all the *prerequisites* to deploy you can go ahead and deploy CommCare HQ! You can follow one of the deployment guides below depending on the type of deployment option that's suitable for you.

1. *Quick Install on Single Server*: This tutorial helps you install CommCare HQ on a single machine with an install wizard. Most of the install is done automatically with configuration set to sensible defaults. This is useful to quickly get started, test or preview what's involved in a working CommCare HQ environment.
2. *Install Using Commcare-Cloud on one or more machines*: This tutorial helps you install CommCare HQ on a single machine or a small cluster using *CommCare Cloud Deployment Tool*, the command-line tool used to not only install CommCare HQ but also to manage a CommCare HQ instance through its entire life-cycle. This method gives you more visibility into installation process, more control and configuration options suitable to your own needs. This is the recommended way to setup a multi machine production grade CommCare HQ instance.

3.1 Quick Install on Single Server

This is a guide on how to deploy a CommCare HQ instance on a monolith server using an install script. Please refer to *Deploy CommCare HQ* guide to decide if this is the right deployment method for you before proceeding.

3.1.1 Prerequisites

- A single Ubuntu 22.04 64-bit server
- Root user to SSH into the server
- git must be installed on the server. If not, please use https://github.com/git-guides/install-git#debianubuntu to install git
- We recommend using Python 3.10 with commcare-cloud. Follow instructions at https://commcare-cloud.readthedocs.io/en/latest/installation/2-manual-install.html#upgrade-to-python-3-10 to upgrade.

3.1.2 Installation Steps

SSH into the server with a root or a user with root privileges, and follow the steps below.

1. Download the commcare-cloud repository.

```
git clone https://github.com/dimagi/commcare-cloud
```

2. Change into the install directory and populate the config file.

```
cd commcare-cloud/quick_monolith_install
# copy the sample config file
cp install-config.yml.sample install-config.yml
# fill the config and save
vim install-config.yml
```

3. Run the installation script. (You may be prompted for sudo password)

```
bash cchq-install.sh install-config.yml
```

3.1.3 Post Installation and Server Administration

Tracking environments directory

On successful installation, the script creates an *environments config directory* under *~/environments* that stores all of the configuration. We recommend that you track this via a version control system such as git. This way you can track changes, and share the directory with other team members who may need to perform server administration using commcare-cloud.

Note: You do not need to track install-config.yml in git as it's only relevant for this installation.

Running commcare-cloud commands

Note that to run any commcare-cloud commands after quick-install you need to login to the VM as either the *ssh_username* configured under *install-config.yml* or as the *ansible* user.

If you wish to let other team members run commcare-cloud commands, you can refer to *User Access Management*.

Once you have installed CommCare HQ successfully you can refer to *First Steps with CommCare HQ* before making your CommCare HQ instance live.

3.1.4 Troubleshooting

The `cchq-install.sh` is an automation of the manual steps listed in *Install Using Commcare-Cloud on one or more machines*. If this script fails before it executes `commcare-cloud $env_name deploy-stack --skip-check --skip-tags=users -e 'CCHQ_IS_FRESH_INSTALL=1' -c local --quiet`, you may rerun the script itself. If the script fails at this or latter commands, you can run those commands one after another instead of re-running the enitre *cchq-install.sh* script to save time. Below are the rest of the commands.

To run the commands below, you need to SSH into the machine as the user added earlier or as ansible user.

```
# $env_name is the name of your environment
commcare-cloud $env_name deploy-stack --skip-check --skip-tags=users -e 'CCHQ_IS_FRESH_
↪INSTALL=1' -c local --quiet
commcare-cloud $env_name django-manage create_kafka_topics
commcare-cloud $env_name django-manage preindex_everything
commcare-cloud $env_name deploy
```

If you have any issues while deploying please refer to *Troubleshooting first time setup*.

3.2 Install Using Commcare-Cloud on one or more machines

This tutorial will walk you through the process of setting up a new environment to run CommCare HQ using *commcare-cloud*. It covers both a single-server ("monolith") environment and a small cluster of virtual machines. If you want to quickly test or preview the environment setup on a single machine you can follow *Quick Install on Single Server* which uses a script to automate all of the below.

This assumes you have gone through *Deploy CommCare HQ* which details what all you need to know to deploy CommCare HQ in production.

3.2.1 Procure Hardware

The first step is to procure the hardware required to run CommCare HQ to meet your project requirements. To understand the hardware resources required for your project please see *Hardware requirements and Deployment Options*. Below are configurations used for the purpose of the tutorial.

Single server

When CommCare HQ is running on a single server, this configuration is referred to as a "monolith". A monolith will need an *absolute minimum* of:

- 4 CPU cores
- 16 GB RAM
- 40 GB storage

These resources are only sufficient to run a demo of CommCare HQ. Any production environment will need a lot more resources.

If you are using VirtualBox for testing CommCare HQ, you can follow the instructions on *Configuring VirtualBox for testing CommCare HQ*.

Cluster

The following example uses a cluster of similarly resourced virtual machines. Let us assume that we have estimated that the following will meet the requirements of our project:

Hostname	vCPUs	RAM	Storage
control1	2	4 GB	30 GB
proxy1	2	4 GB	30 GB
webworker1	2	8 GB	30 GB
webworker2	2	8 GB	30 GB
db1	2	16 GB	30 GB + 60 GB
db2	2	16 GB	30 GB + 60 GB + 20 GB

db1 has an extra volume for databases. db2 has one extra volume for databases, and another for a shared NFS volume.

All environments

CommCare HQ environments run on Ubuntu Server 22.04 (64-bit).

During the installation of Ubuntu you will be prompted for the details of the first user, who will have sudo access. It is convenient to name the user "ansible". (The user can be named something else. Deploying CommCare HQ will create an "ansible" user if one does not already exist.)

When choosing which software to install during the Ubuntu installation, select only "SSH Server".

You will need a domain name which directs to the monolith or the cluster's proxy server.

3.2.2 Prepare all machines for automated deploy

Do the following on the monolith, or on each machine in the cluster.

Enable root login via SSH

On a standard Ubuntu install, the root user is not enabled or allowed to SSH. The root user will only be used initially, and will then be disabled automatically by the install scripts.

Make a root password and store it somewhere safe for later reference.

1. Set the root password:

```
$ sudo passwd root
```

2. Enable the root user:

```
$ sudo passwd -u root
```

3. Edit `/etc/ssh/sshd_config`:

```
$ sudo nano /etc/ssh/sshd_config
```

   To allow logging in as root, set

```
PermitRootLogin yes
```

   To allow password authentication, ensure

```
PasswordAuthentication yes
```

4. Restart SSH:

```
$ sudo service ssh reload
```

Initialize log file

To be used in the installation process.

```
$ sudo touch /var/log/ansible.log
$ sudo chmod 666 /var/log/ansible.log
```

Install system dependencies

This only needs to be done on the control machine. In the case of a monolith, there is only one machine to manage so that is also the control machine. In our example cluster, the control machine is named "control1".

1. SSH into control1 as the "ansible" user, or the user you created during installation. You can skip this step if you are installing a monolith:

   ```
   $ ssh ansible@control1
   ```

   This instruction assumes that the control machine's name resolves to its IP address. Replace the name with the IP address if necessary.

2. On the control machine, or the monolith, install required packages:

   ```
   $ sudo apt update
   $ sudo apt install python3-pip python3-dev python3-distutils python3-venv libffi-
   ↪dev sshpass net-tools
   ```

3. Check your default Python version for Python 3.x:

   ```
   $ python --version
   ```

   If your default version is not 3.x or if the "python" command was not found, make python3 your default by running the command below, otherwise skip it.

   ```
   $ sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 10
   ```

3.2.3 Create a user for yourself

In general, CommCare environments are managed by a team. Each member of the team has their own user account.

On the control machine or the monolith, create a user for yourself, and add them to the "sudo" user group. For example, if your username were "jbloggs", the commands would be

```
$ sudo adduser jbloggs
...
$ sudo usermod -a -G sudo jbloggs
```

3.2.4 Configure SSH

If you do not have an SSH key pair already, you will need to create one. (Substitute "jbloggs@example.com" with your email address)

```
$ ssh-keygen -t rsa -b 4096 -C "jbloggs@example.com"
```

Cluster only: Copy an SSH key pair for your user to the control machine. For example, if the key pair you want to copy is `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`, then the commands to copy the SSH key pair would be

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub jbloggs@control1
$ scp ~/.ssh/id_rsa{,.pub} control1:.ssh/
```

You can now log in using your SSH key:

```
(jbloggs@jbloggs-pc) $ ssh control1
```

3.2.5 Install CommCare Cloud

1. On the control machine or the monolith, install and configure Git:

   ```
   $ sudo apt install git
   $ git config --global user.name "Jay Bloggs"
   $ git config --global user.email "jbloggs@example.com"
   ```

   (Of course, substitute "Jay Bloggs" with your name, and "jbloggs@example.com" with your email address.)

2. Clone and initialize CommCare Cloud:

   ```
   $ git clone https://github.com/dimagi/commcare-cloud.git
   $ cd commcare-cloud
   $ source control/init.sh
   ```

   When prompted, confirm setting up the CommCare Cloud environment on login:

   ```
   Do you want to have the CommCare Cloud environment setup on login?
   (y/n): y
   ```

3. Clone the sample CommCare Cloud "environments" folder into your home directory.

   ```
   $ cd ~
   $ git clone https://github.com/dimagi/sample-environment.git environments
   ```

4. Rename your environment. You could name it after your organization or your project. If you are installing a monolith you could leave its name as "monolith". For this example we will name it "cluster".

   ```
   $ cd environments
   $ git mv monolith cluster
   $ git commit -m "Renamed environment"
   ```

5. Remove the "origin" Git remote. (You will not be pushing your changes back to the Dimagi "sample-environment" repository.)

   ```
   $ git remote remove origin
   ```

6. (Optional) You are encouraged to add a remote for your own Git repository, so that you can share and track changes to your environment's configuration. For example:

```
$ git remote add origin git@github.com:your-organization/commcare-environment.git
```

7. Configure your CommCare environment.

   See *Configuring your CommCare Cloud Environments Directory* for more information.

8. Add your username to the `present` section of `~/environments/_users/admins.yml`.

```
$ nano ~/environments/_users/admins.yml
```

9. Copy your **public** key to `~/environments/_authorized_keys/`. The filename must correspond to your username.

   For example:

```
$ cp ~/.ssh/id_rsa.pub ~/environments/_authorized_keys/$(whoami).pub
```

10. Change "monolith.commcarehq.test" to your real domain name,

```
$ cd cluster
```

    (or whatever you named your environment, if not "cluster".)

```
$ git grep -n "monolith"
```

    You should find references in the following files and places:

    - `proxy.yml`
      - `SITE_HOST`
    - `public.yml`
      - `ALLOWED_HOSTS`
      - `server_email`
      - `default_from_email`
      - `root_email`

11. Configure `inventory.ini`

For a monolith

1. Find the name and IP address of the network interface of your machine, and note it down. You can do this by running

```
$ ip addr
```

   This will give an output that looks similar to

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group␣
↪default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
```

(continues on next page)

(continued from previous page)

```
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP␣
↪group default qlen 1000
    link/ether 08:00:27:48:f5:64 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 85228sec preferred_lft 85228sec
    inet6 fe80::a00:27ff:fe48:f564/64 scope link
       valid_lft forever preferred_lft forever
```

Here, the network interface we are interested in is **enp0s3**, which has an IP address of `10.0.2.15`. Note this address down.

2. Open your environment's `inventory.ini` file in an editor. (Substitute "cluster".)

```
$ nano ~/environments/cluster/inventory.ini
```

Replace the word `localhost` with the IP address you found in the previous step.

Uncomment and set the value of `ufw_private_interface` to the network interface of your machine.

For a cluster

Having planned and provisioned the virtual machines in your cluster, you will already know their hostnames and IP addresses.

The following is an example of an `inventory.ini` file for a small cluster. Use it as a template for your environment's `inventory.ini` file:

```
[proxy1]
192.168.122.2 hostname=proxy1 ufw_private_interface=enp1s0

[control1]
192.168.122.3 hostname=control1 ufw_private_interface=enp1s0

[webworker1]
192.168.122.4 hostname=webworker1 ufw_private_interface=enp1s0

[webworker2]
192.168.122.5 hostname=webworker1 ufw_private_interface=enp1s0

[db1]
192.168.122.4 hostname=db1 ufw_private_interface=enp1s0 elasticsearch_node_name=es0␣
↪kafka_broker_id=0

[db2]
192.168.122.5 hostname=db1 ufw_private_interface=enp1s0 elasticsearch_node_name=es1␣
↪kafka_broker_id=1

[control:children]
control1
```

(continues on next page)

(continued from previous page)

```
[proxy:children]
proxy1

[webworkers:children]
webworker1
webworker2

[celery:children]
webworker1
webworker2

[pillowtop:children]
webworker1
webworker2

[django_manage:children]
webworker1

[formplayer:children]
webworker2

[rabbitmq:children]
webworker1

[postgresql:children]
db1
db2

[pg_backup:children]
db1
db2

[pg_standby]

[couchdb2:children]
db1
db2

[couchdb2_proxy:children]
db1

[shared_dir_host:children]
db2

[redis:children]
db1
db2

[zookeeper:children]
db1
db2
```

(continues on next page)

(continued from previous page)

```
[kafka:children]
db1
db2

[elasticsearch:children]
db1
db2
```

12. Configure the `commcare-cloud` command.

```
$ export COMMCARE_CLOUD_ENVIRONMENTS=$HOME/environments
$ manage-commcare-cloud configure
```

You will see a few prompts that will guide you through the installation. Answer the questions as follows for a standard installation. (Of course, substitute "jbloggs" with your username, and "cluster" with the name of your environment.)

```
Do you work or contract for Dimagi? [y/N] n

I see you have COMMCARE_CLOUD_ENVIRONMENTS set to /home/jbloggs/environments in␣
↪your environment
Would you like to use environments at that location? [y/N] y
```

As prompted, add the commcare-cloud config to your profile to set the correct paths:

```
$ echo "source ~/.commcare-cloud/load_config.sh" >> ~/.profile
```

Load the commcare-cloud config so it takes effect immediately:

```
$ source ~/.commcare-cloud/load_config.sh
```

Copy the example config file:

```
$ cp ~/commcare-cloud/src/commcare_cloud/config.example.py ~/commcare-cloud/src/
↪commcare_cloud/config.py
```

Update the known hosts file

```
$ commcare-cloud cluster update-local-known-hosts
```

13. Generate secured passwords for the vault

In this step, we'll generate passwords in the `vault.yml` file. This file will store all the passwords used in this CommCare environment. (Once again, substitute "cluster" with the name of your environment.)

```
$ python ~/commcare-cloud/commcare-cloud-bootstrap/generate_vault_passwords.py --
↪env='cluster'
```

Before we encrypt the `vault.yml` file, have a look at the `vault.yml` file. (Substitute "cluster".)

```
$ cat ~/environments/cluster/vault.yml
```

Find the value of "ansible_sudo_pass" and record it in your password manager. We will need this to deploy CommCare HQ.

14. Encrypt the provided vault file, using that "ansible_sudo_pass". (As usual, substitute "cluster" with the name of your environment.)

```
$ ansible-vault encrypt ~/environments/cluster/vault.yml
```

More information on Ansible Vault can be found in the Ansible help pages.

Managing secrets with Vault will tell you more about how we use this vault file.

3.2.6 Deploy CommCare HQ services

You will need the SSH agent to have your SSH key for Ansible.

```
$ eval `ssh-agent`
$ ssh-add ~/.ssh/id_rsa
```

When you run the "commcare-cloud deploy-stack", you will be prompted for the vault password from earlier. You will also be prompted for an SSH password. This is the root user's password. After this step, the root user will not be able to log in via SSH.

```
$ commcare-cloud cluster deploy-stack --first-time -e 'CCHQ_IS_FRESH_INSTALL=1'

This command will apply without running the check first. Continue? [y/N]y
ansible-playbook /home/jbloggs/commcare-cloud/src/commcare_cloud/ansible/deploy_stack.
↪yml -i /home/jbloggs/environments/cluster/inventory.ini -e @/home/jbloggs/environments/
↪cluster/vault.yml -e @/home/jbloggs/environments/cluster/public.yml -e @/home/jbloggs/
↪environments/cluster/.generated.yml --diff --tags=bootstrap-users -u root --ask-pass --
↪vault-password-file=/home/jbloggs/commcare-cloud/src/commcare_cloud/ansible/echo_vault_
↪password.sh --ask-pass --ssh-common-args -o=UserKnownHostsFile=/home/jbloggs/
↪environments/cluster/known_hosts
Vault Password for 'cluster': <ansible_sudo_pass>
SSH password: <root user's password>
```

This will run a series of Ansible commands that will take quite a long time to run.

If there are failures during the install, which may happen due to timing issues, you can continue running the playbook with:

```
$ commcare-cloud cluster deploy-stack --skip-check -e 'CCHQ_IS_FRESH_INSTALL=1'
```

3.2.7 Deploy CommCare HQ code

Deploying CommCare HQ code for the first time needs a few things set up initially.

1. Create Kafka topics:

```
$ commcare-cloud cluster django-manage create_kafka_topics
```

2. Create the CouchDB and Elasticsearch indices:

```
$ commcare-cloud cluster django-manage preindex_everything
```

3. Run the "deploy" command:

```
$ commcare-cloud cluster deploy
```

Or if you need to deploy a specific version of CommCare HQ as opposed to the latest:

```
$ commcare-cloud cluster deploy --commcare-rev=<commit-hash>
```

When prompted for the `sudo` password, enter the "ansible_sudo_pass" value.

See the Deploying CommCare HQ code changes section in *Managing The Deployment* for more information.

If deploy fails, you can restart where it left off:

```
$ commcare-cloud cluster deploy --resume
```

3.2.8 Set up valid SSL certificates

1. Run the playbook to request a Let's Encrypt certificate:

   ```
   $ commcare-cloud cluster ansible-playbook letsencrypt_cert.yml --skip-check
   ```

2. Update settings to take advantage of new certs:

   ```
   $ nano $COMMCARE_CLOUD_ENVIRONMENTS/cluster/proxy.yml
   ```

   and set `fake_ssl_cert` to `False`

3. Deploy proxy again

   ```
   $ commcare-cloud cluster ansible-playbook deploy_proxy.yml --skip-check
   ```

3.2.9 Clean up

CommCare Cloud will no longer need the root user to be accessible via the password. The password can be removed if you wish, using

```
$ sudo passwd -d -l root
```

3.2.10 Test and access CommCare HQ

Testing your new CommCare Environment

Run the following command to test each of the backing services as described 'Checking services once deploy is complete' section in *Managing The Deployment*.

```
$ commcare-cloud cluster django-manage check_services
```

Following this initial setup, it is also recommended that you go through this *Testing your new CommCare Environment*, which will exercise a wide variety of site functionality.

Accessing CommCare HQ from a browser

If everything went well, you should now be able to access CommCare HQ from a browser.

If you are using VirtualBox, see *Configuring VirtualBox for testing CommCare HQ* to find the URL to use in your browser.

Troubleshooting first-time setup

If you face any issues, it is recommended to review the *Troubleshooting first time setup* documentation.

Firefighting issues once CommCare HQ is running

You may also wish to look at the firefighting page which lists out common issues that `commcare-cloud` can resolve.

If you ever reboot this machine, make sure to follow the *after reboot procedure* in the firefighting doc to bring all the services back up, and mount the encrypted drive by running:

```
$ commcare-cloud cluster after-reboot all
```

3.2.11 First Steps with CommCare HQ

If you are migrating data you can refer to *Migrate a Project from one instance to a new instance* or *Migrating an entire CommCare HQ instance*. Otherwise, you can do below to start using CommCare HQ.

Make a user

If you are following this process, we assume you have some knowledge of CommCare HQ and may already have data you want to migrate to your new cluster. By default, the deploy scripts will be in `Enterprise` mode, which means there is no sign up screen. You can change this and other settings in the localsettings file by following the *localsettings deploy instructions* in *Managing The Deployment*.

If you want to leave this setting as is, you can make a superuser with:

```
$ commcare-cloud cluster django-manage make_superuser {email}
```

where `{email}` is the email address you would like to use as the username.

Note that promoting a user to superuser status using this command will also give them the ability to assign other users as superuser in the in-app Superuser Management page.

Add a new CommCare build

In order to create new versions of applications created in the CommCare HQ app builder, you will need to add the the latest `CommCare Mobile` and `CommCare Core` builds to your server. You can do this by running the command below - it will fetch the latest version from GitHub.

```
$ commcare-cloud cluster django-manage add_commcare_build --latest
```

Link to a project on other CommCare HQ instance

If you intend to use Linked Projects feature to link projects on between two different instances of CommCare HQ, you may refer to Remote Linked Projects to set this up.

3.2.12 Operations

Once you have your CommCare HQ live, please refer to *Operations and maintenance* for maintaining your environment.

To add new server administrators please refer to *Setting up CommCare HQ Server Administrators*.

3.3 Troubleshooting first time setup

3.3.1 My site is showing "Internal Server Error"

If you are seeing a blank screen with just the words "Internal Server Error" on it, it means that the django webworker process is not reporting as "down", but still failing to bootstrap fully. (If you are seeing a more elaborate 500 page, then that is an issue with a single request, but ususally does not indicate a more pervasive problem with the site's ability to receive and handle requests in general.) Often this is because it is unable to connect to a backing service (such as CouchDB, Redis, PostgreSQL, PgBouncer, Elasticsearch, Riak). This in turn can fall into a number of categories of issue:

1. the service is down
2. the service is up, but unreachable due to network configuration
3. the service is up and reachable, but is blocking the connection for a permissions-related reason (auth is wrong, IP is not whitelisted, etc.)
4. the service is up, reachable, and accepting the connection, but is failing due to other problems such as misconfiguration (e.g. a low max connection limit), or other problem such as out of memory errors.

Further diagnosis

You can start by simply checking which of the backing services the application code is able to connect to by running

```
commcare-cloud <env> django-manage check_services
```

Note that this checks the availability of each service *to the application code*, so it could be any type of problem given in 1-4 above.

Steps to fix

If a stack trace you find in the logs points at a service being down, you can check its status and start it if it's stopped or restart it if it's "up" but causing problems. In the command below replace the word "postgresql" with the name of the service at issue:

```
commcare-cloud <env> service postgresql status
commcare-cloud <env> service postgresql start  # to start it or
commcare-cloud <env> service postgresql restart  # to restart it
```

Digging into the problem

If that doesn't fix it, you will need to dig a bit deeper.

Start by checking which of the application services are reporting as up by running

```
commcare-cloud <env> service commcare status
```

You will likely find the `django` process is reporting as RUNNING. Some other processes if affected by a similar issue may (or may not) be reporting as FATAL "exited too quickly".

To dig into a particular error, you can log into the machine and tail one of the logs:

```
commcare-cloud <env> ssh webworkers[0]
$ tail -n100 /home/cchq/www/<env>/log/django.log
```

or, if you do not want to figure out where a particular log lives, you can run the command on all machines (allowing that it'll fail on any machine that doesn't contain that particular log):

```
commcare-cloud <env> run-shell-command all 'tail -n100 /home/cchq/www/<env>/log/django.
↪log'
```

or, you can use the output from the status command above and run it through the `supervisorctl` command:

```
commcare-cloud <env> ssh <machine>
$ sudo supervisorctl tail -f <supervisor process name>
```

3.3.2 One of the setup commands is showing...

`RequestError: socket.error: [Errno 111] Connection refused`

This means that CouchDB is unreachable.

Breakdown of a request to CouchDB

Note: if you are running on a recommended single-machine setup, then you can ignore the host groups (denoted `[in brackets]`): all services will be running on the same machine.

Requests to CouchDB are made over HTTP, and are normally routed the following way:

1. They start at the originator of the request, such as a Django web worker
2. They are made to port 25984 on host `[couchdb_proxy]`, which is served by the `nginx` web server, acting as a load balancer.
3. `nginx` passes them through to one of the `couchdb2` nodes (or *the* `couchdb2` node if you have only one), which handles the requests.

```
[webworkers] [couchdb2_proxy] [couchdb2]
django  -->  nginx  -------->  couchdb2
             port 25984        port 15984
```

The following table represents the general case and includes variables that may be overriding the default port values:

	host group	service	port (default value)	port (variable name)
Originator	various	various		
CouchDB Load Balancer	`[couchdb2_proxy]`	`nginx`	25984	`couchdb2_proxy_port`
CouchDB Node	`[couchdb2]`	`couchdb2`	15984	`couchdb2_port`

How to confirm the issue

To confirm the issue, that django processes cannot reach CouchDB, run

```
commcare-cloud <env> django-manage check_services couch
```

It should tell you that CouchDB is unreachable.

How to solve

The first thing to check is whether couchdb2 and couchdb2_proxy services are up, which you can do with the single command:

```
commcare-cloud <env> service couchdb2 status
```

If one of the services is reporting down, you can use the following to start it:

```
# Start both
commcare-cloud <env> service couchdb2 start

# or start only couchdb2
commcare-cloud <env> service couchdb2 start --only couchdb2

# or start only couchdb2_proxy
commcare-cloud <env> service couchdb2 start couchdb2_proxy
```

If CouchDB is still unreachable, try hitting each of the individual parts.

1. Test whether `couchdb2` is responding .. code-block:: bash

   commcare-cloud <env> ssh couchdb2 curl <couchdb2-internal-IP-address>:15984

2. Test whether the load balancer on `couchdb2_proxy` is responding .. code-block:: bash

   commcare-cloud <env> ssh couchdb2_proxy curl <couchdb2_proxy-internal-IP-address>:25984

Notes:

- You will often see the value for `<couchdb2-internal-IP-address>` printed out next to `eth0` upon `ssh`ing into the machine.
- For a single-machine setup, no need to separately ssh for each step.

Is the CouchDB `nginx` site on `couchdb2_proxy` enabled?

```
commcare-cloud <env> ssh ansible@couchdb2_proxy
ls /etc/nginx/sites-enabled
```

This should contain a file with "couchdb" in the name.

Are there errors in the `couchdb2` logs?

```
commcare-cloud <env> ssh ansible@couchdb2
ls /usr/local/couchdb2/couchdb/var/log/
```

There should be some logs in there that you can tail or grep through for errors.

3.3.3 One of the setup commands is showing…

`Error requesting archive. Problem with NPM phantomjs package downloading and path not found`

```
ErrorMessage:
Status: 404
Request options: {
    "url": "https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-1.9.8-linux-x86_64.
↪tar.bz2"
    "encoding": null,
    "followRedirect": true,
    "headers": {},
    "strictSSL": true
}
```

steps to resolve

```
cd /usr/local/share
sudo wget https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-1.9.8-linux-x86_64.
↪tar.bz2
sudo tar xjf phantomjs-1.9.8-linux-x86_64.tar.bz2
sudo ln -s /usr/local/share/phantomjs-1.9.8-linux-x86_64/bin/phantomjs /usr/local/share/
↪phantomjs
sudo ln -s /usr/local/share/phantomjs-1.9.8-linux-x86_64/bin/phantomjs /usr/local/bin/
↪phantomjs
sudo ln -s /usr/local/share/phantomjs-1.9.8-linux-x86_64/bin/phantomjs /usr/bin/phantomjs
```

3.4 Testing your new CommCare Environment

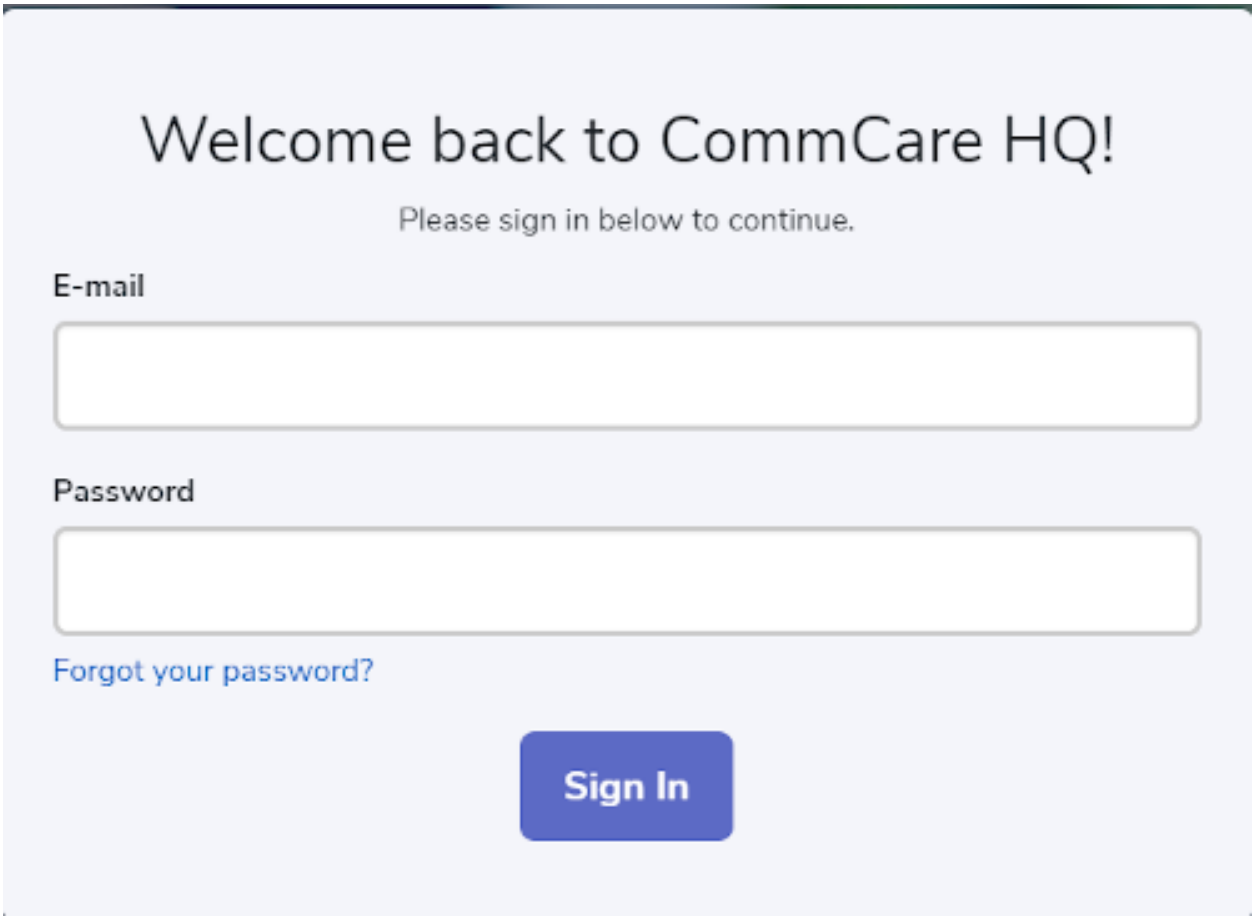
This document can be used to ensure that your self-hosted CommCare environment is working as expected by running through key CommCare HQ processes. We'll go through these routines and processes step-by-step and when complete, you can be confident that your CommCare instance is set up correctly. In total, this plan will take about one hour to execute.

If any of these steps fail, we advise troubleshooting by first evaluating the related processes detailed at the end of each step. You'll find some specifics on the *Firefighting Production Issues*.

3.4.1 Step 1: Logging in

Simply logging into CommCare HQ tests multiple processes.

Begin by logging into your application with your username and password.



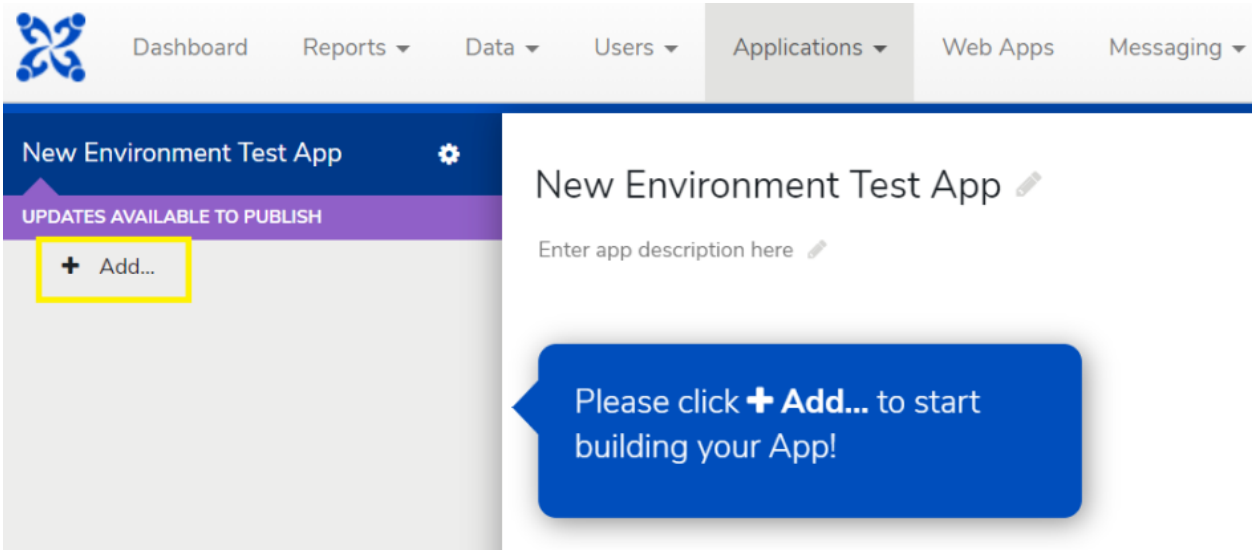
This step tests: that Django, Postgres, Couch, and Redis are functional. Furthermore, it ensures that staticfiles are being served correctly.

3.4.2 Step 2a: Creating an application

In this step, we will create a basic Case Management app, populate it with questions and add case properties.

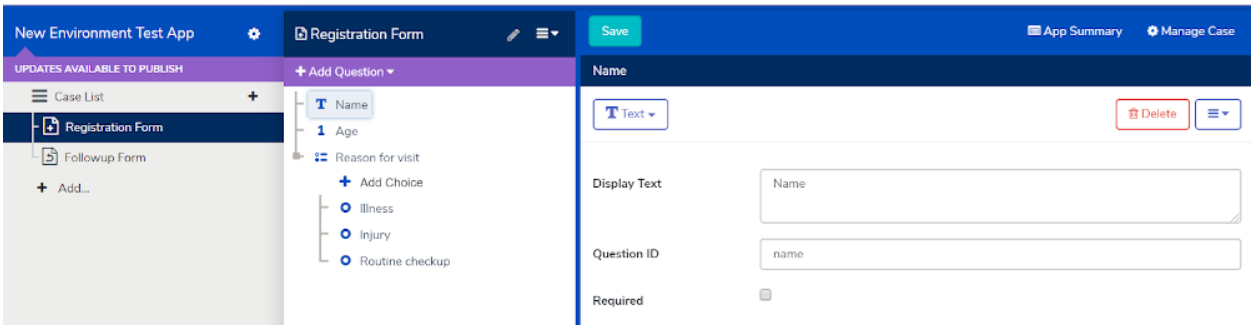
We'll create an application by selecting either of the Application links from the Dashboard page; the Dashboard page is the landing page post login. There is one Application link on the top ribbon and another in the box on the screen with a phone icon.

You'll be brought to the page below. Give your app a name and click the '+Add' button. Choose the option 'Case List' to create an application that uses Case Management. We'll use case management to create cases later on.



3.4.3 Step 2b: Populating the Registration Form

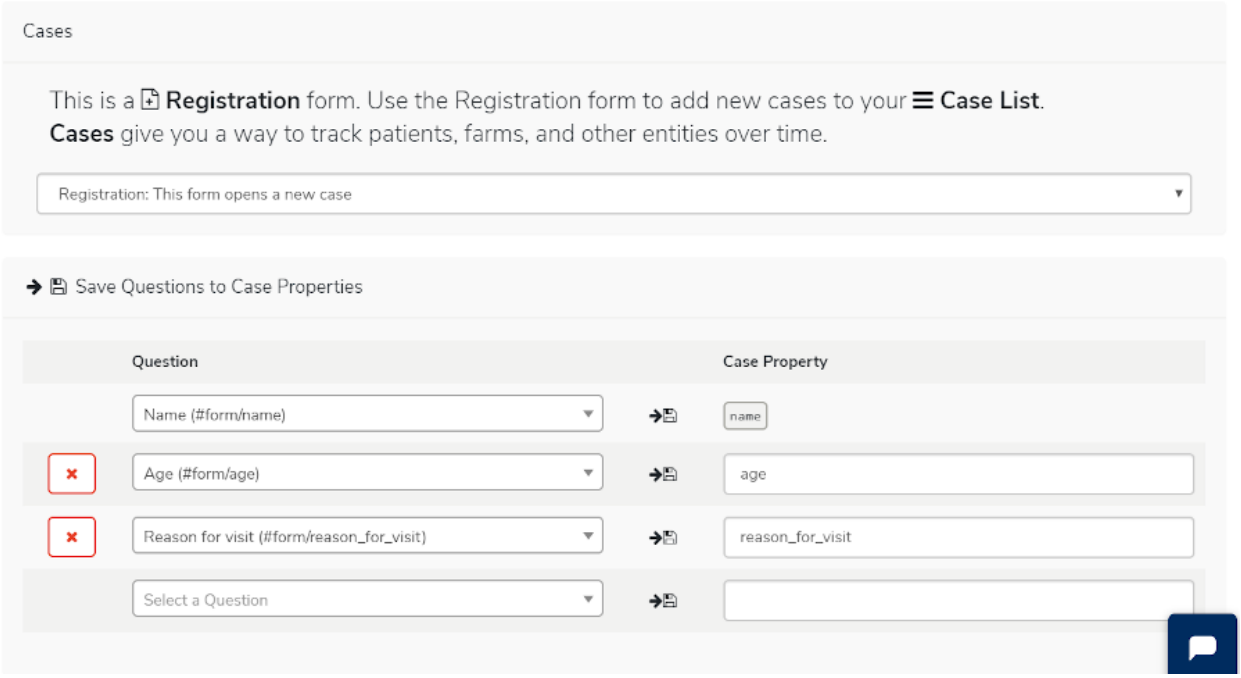
The previous step will create two forms under your case list and you'll be navigated to a page called the 'Formbuilder.' You can populate these forms with questions by selecting the purple '+Add Question' button. Let's start by populating the Registration Form with a few questions. Feel free to use the template I've included in the image below or create your own questions. If using your own questions, do use one text question to capture names so we can easily refer to our cases later.



Once questions have been added, save and then click the 'Manage Case' button at the top right of the screen. This will navigate to the Registration Form's Case Management tab.

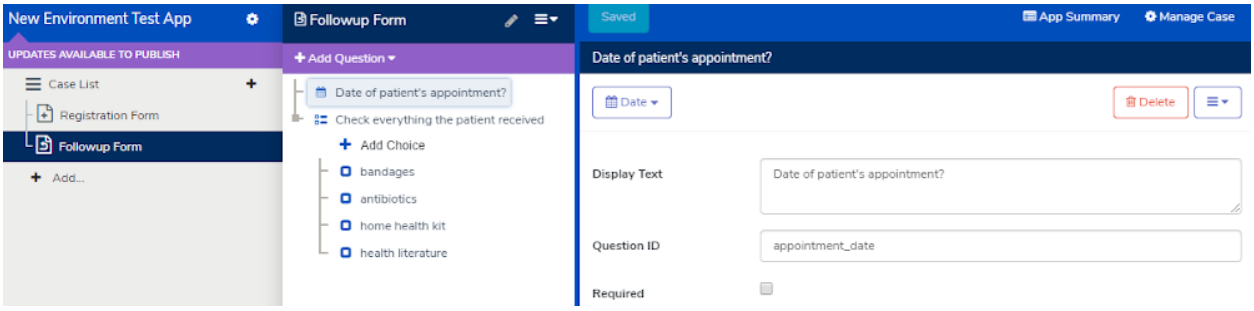
3.4.4 Step 2c: Saving Case Properties

In this section, add your questions as Case Properties by selecting them in the dropdowns under the 'Save Questions to Case Properties' section. Saving a question as a Case Property will allow that information to be associated with the case. We'll follow up on these case properties later once we begin using our app.



3.4.5 Step 2d: Populating and creating properties for the Followup Form

Repeat this process with the Followup Form. You can click on the 'Followup Form' text on the left of the screen and this will navigate you back to the Formbuilder. Again, add a few questions of your own, or use my template below. Once complete, save and select 'Manage Case.' Then, add those questions as Case Properties and save again.

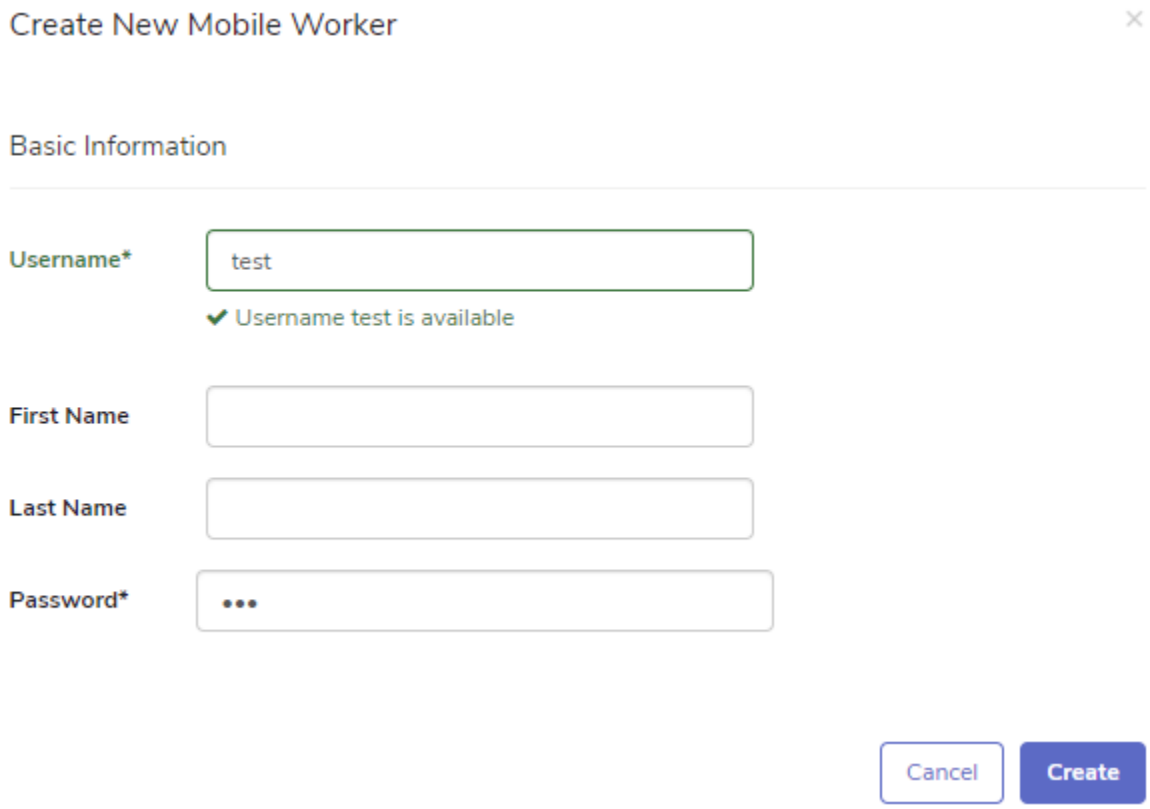


These steps test: that Applications can be created. Applications are the core of how CommCare works and will be used to collect data for your project. The App Building process also validates that Formplayer is functional.

To further validate Formplayer, click the teal bar on the right of the screen, this will open a tool called Application Preview. If Formplayer isn't running or isn't connected properly, the Application Preview pane will be blank.

3.4.6 Step 3: Creating a mobile worker

We'll create a mobile worker to access our application and submit data. To do so, select Users from the top ribbon and click Mobile Workers. A new page should load and you should then see a blue button '+Create Mobile Worker,' click this and the popup below will appear. Enter whatever username and password combo you'd like.



Your mobile worker should now appear as a link under the 'New Mobile Workers' section.

This step tests: Creating a mobile worker and validating a phone number. The page that lists mobile workers relies on Elasticsearch.

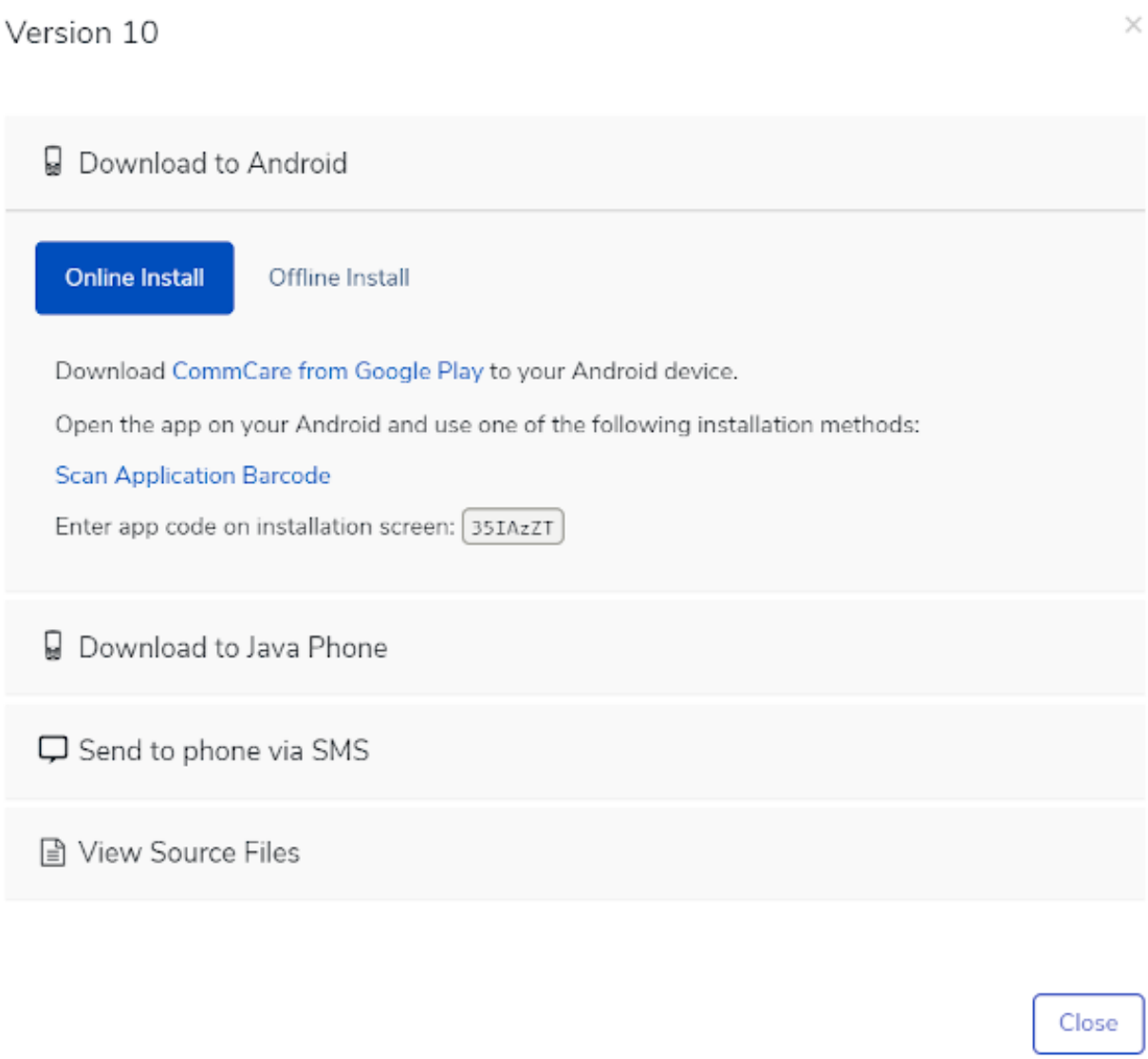
3.4.7 Step 4a: Deploying your application and submitting forms from mobile

Note: You may skip this step if your project intends to use only Web Apps for data collection.

Now that we've built an app, we'll validate its deploy. In the top ribbon, select Applications and click your newly created test application. This will navigate you to the Releases page. Click the blue 'Make New Version' button and within a few moments, a version will be available.

Before proceeding, you'll need an Android device with the CommCare mobile app installed. If you do not have CommCare installed on your device, first go to the Play Store and download it.

On the 'Releases' page, click the 'Publish' button with your newly created version. This will launch a modal similar to the one in the image below. Your version number may differ from mine:



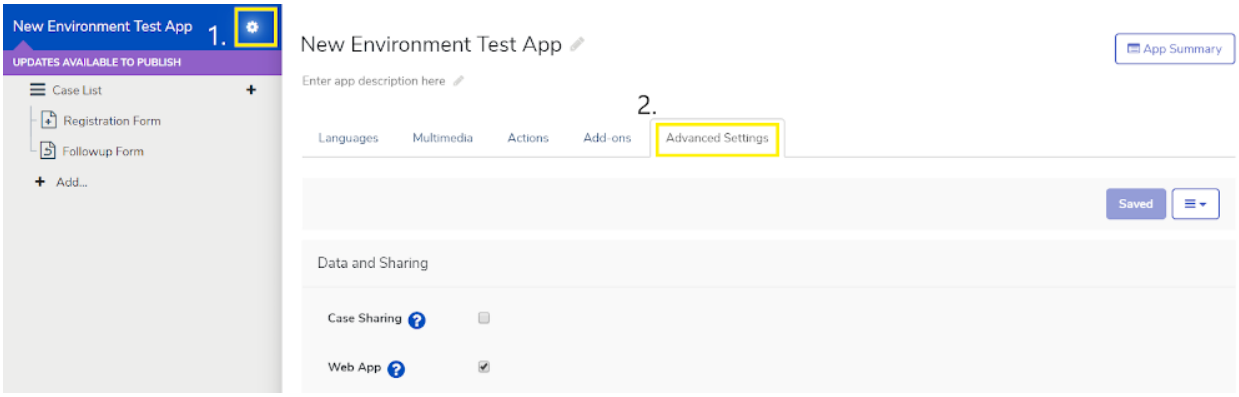
While there are multiple ways to deploy your application to a mobile device, choose 'Scan Application Barcode' to display a QR code. On your mobile device, press 'Scan Application Barcode' to install your application. This should only take a few moments and when complete, you will be able to log in on your mobile device with the mobile worker created in the previous step.

Remaining on your mobile device, access and submit your registration form to create a case and then the followup form to add additional properties to that case. We'll verify this case was successfully created in CommCare HQ shortly. After submitting both the Registration and Followup form, press the blue 'Sync with Server' button on the CommCare mobile app's home screen. This will ensure the forms on your device are pushed to CommCare HQ.

This step tests: When you attempt to build the app, you may see an error "Unable to validate form", this likely indicates that Formplayer isn't running or isn't connected properly to CommCare HQ. The act of logging in on your mobile device for the first time automatically syncs with the server. Logging in validates that mobile devices are able to connect with the server.

3.4.8 Step 4b: Submitting forms through Web Apps

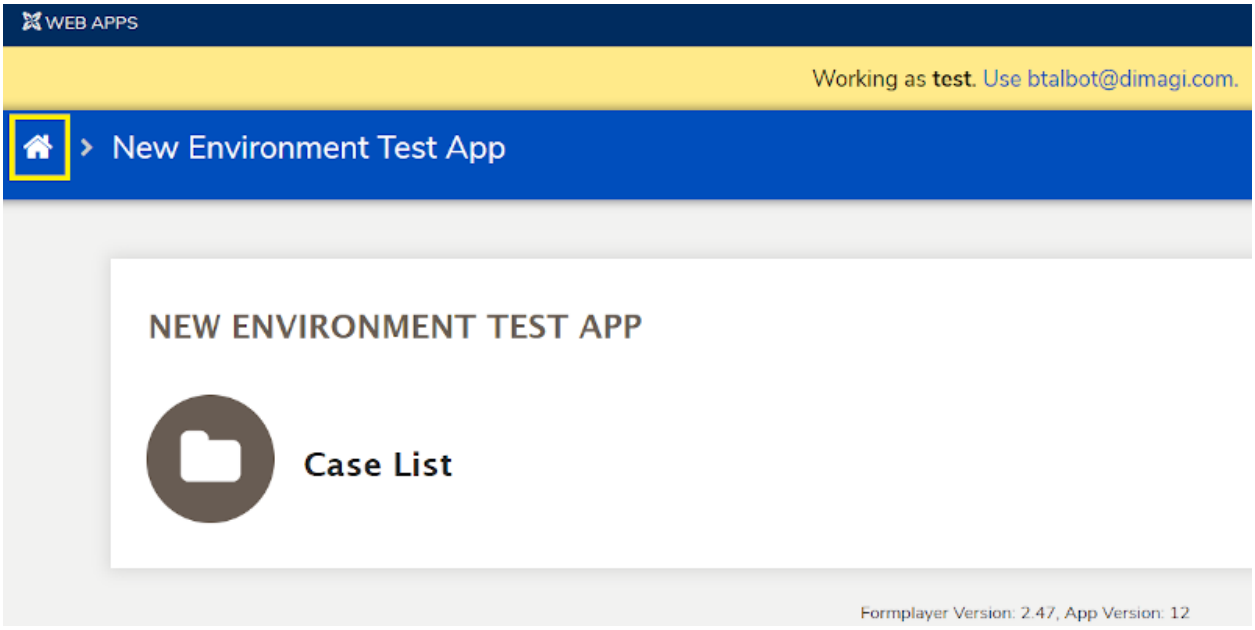
'Web Apps' is a method of submitting data to CommCare HQ online and with a desktop or laptop. Web Apps must be enabled separately under your application's Advanced Settings. Return to CommCare HQ and then click the gear icon next to your application's name. Next, click 'Advanced Settings' to enable Web Apps and save. Upon completing this step, a purple banner will appear under your application's name.



We will need to make a new version of your application on the release page. To do so, click your application's name and the 'Make New Version' button. On this newly created version, it's important to toggle the 'Released/In Test' button to 'Released.'

After marking your version as released, click 'Web Apps' in the top ribbon of CommCare HQ. There, you'll notice a purple button labeled, 'Login As.' Click that and choose your Mobile Worker. While you can submit forms as a Web User (your CommCare HQ login), most CommCare users structure their apps around submissions from Mobile Workers. Using 'Login As' allows you to submit forms as a mobile worker via Web Apps.

Click on your application and submit the registration and follow-up forms. Once the forms have been submitted, click the Home icon in Web Apps and then click the blue Sync button.



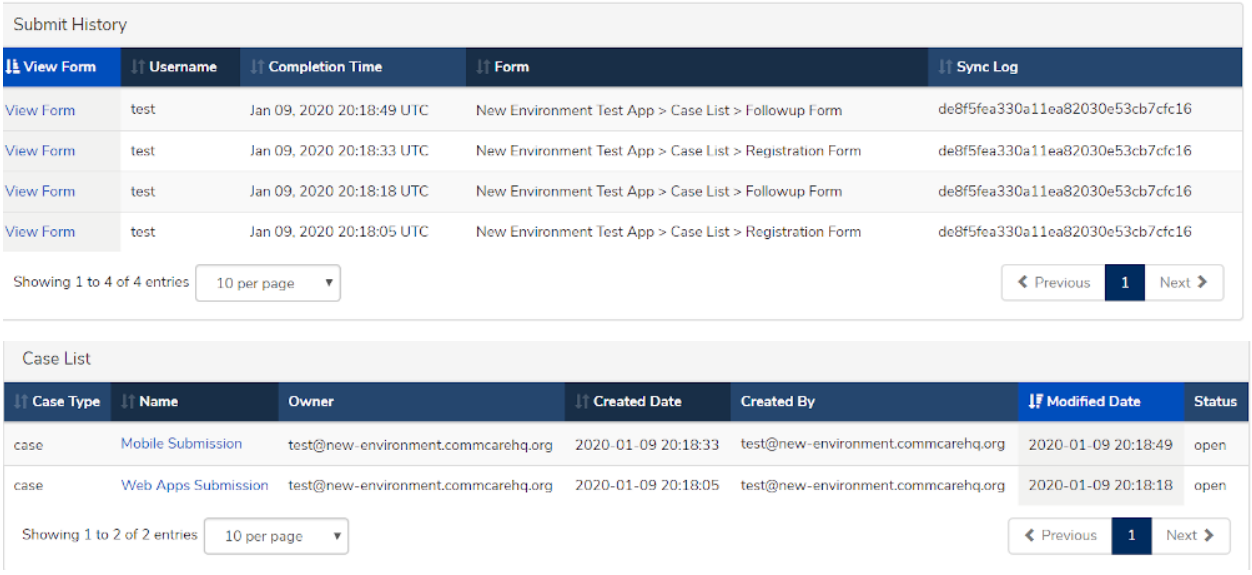
This step tests: Using Web Apps tests that Formplayer is working correctly. We will know whether our submission was a success when we check reports in the next step.

3.4.9 Step 5: Viewing submitted data in reports

In this step, we will run a few key CommCare HQ reports to ensure data was received and accessible.

We've now submitted data from either a mobile device, Web Apps or both. Let's now view this data in CommCare HQ. If still in Web Apps, click 'Show Full Menu' at the top of the screen to view the ribbon. Otherwise, simply access CommCare HQ and click 'Reports', then 'Submit History.'

On the Submit History report, simply accept the defaults and click the blue Apply button. You'll be shown all the forms submitted for your project along with some key pieces of information about those forms. To further test reports, access the Case List report (it should be directly under Submit History on the left of the screen). Again, accept the default filters and click apply.



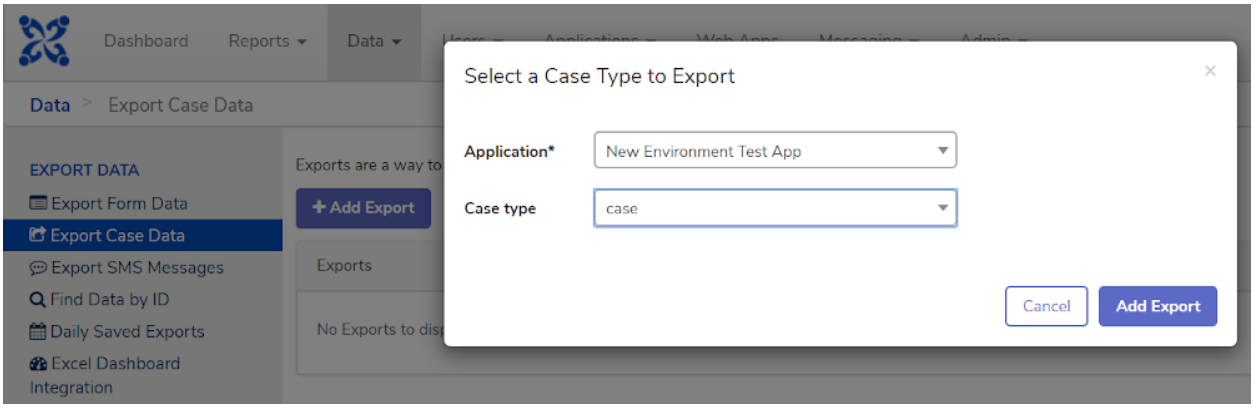
Your results will be unique to your submissions and cases. Feel free to explore these reports further by clicking either 'View Form' or the name of the case.

This step tests: That forms were processed and then picked up by Pillowtop and transferred to the appropriate database. The report's list uses Elasticsearch and its detail use PSQL.

3.4.10 Step 6a: Exporting CommCare Data: Case Export

CommCare HQ offers multiple ways to retrieve and export your data. This section will highlight two of these methods; a standard case export and the creation of an OData feed, which can be used to hook up CommCare HQ to Tableau and Power BI.

A standard case export can be done by clicking 'Data' from the top ribbon on CommCare HQ and choosing 'Export Case Data.' Then, click the '+Add Export' button and define the Application and Case Type in the modal that appears. Though the application name may differ, your screen will look like:



Select 'Add Export,' which will navigate you to the Create Case Data Export page. For now, accept the defaults and click 'Create' at the bottom of the screen. This will return you to the 'Export Case Data' page and next to your newly created export, select the 'Export' button. You'll now be on the Download Case Data Export page, click 'Prepare Export' and when ready, click 'Download' to ensure the export works correctly.

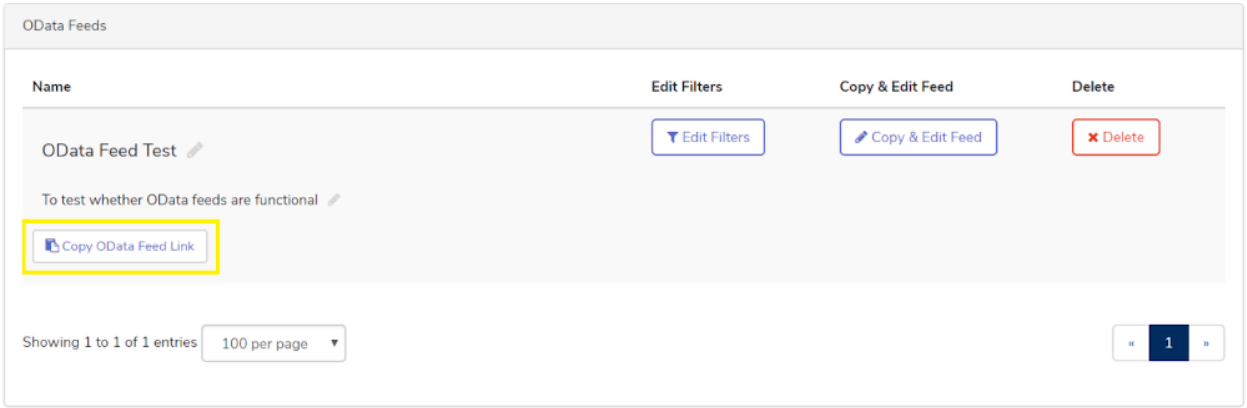
This will initiate an xlsx download of your case data. Feel free to open it and explore, but simply downloading the data is enough to know the routine is functional.

This step tests: Case Exports, and exports more generally, ensure that Celery, BlobDB, and Redis are working as expected. Celery processes the export and saves the result in BlobDB, while process status (the progress bar) is stored in Redis.

3.4.11 Step 6b: (Optional) Exporting CommCare Data: OData Feed

CommCare HQ uses multiple API endpoints to allow users to access their data. One of the APIs built into the CommCare UI is the OData Feed export that can be integrated with Power BI and Tableau. If applicable to your project, let's use this to further ensure our access to CommCare data is sound.

To begin, select Data from the top ribbon and click the Power BI/Tableau Integration. Once the page loads, click the '+Add OData Feed' button. Similar to the Case Export routine we ran earlier, a modal will pop up with a series of prompts. As the Feed Type, choose 'Case,' then select your application and the case type (likely, this will have defaulted to 'case'). Then click 'Add OData Feed' and you'll land on the Create OData Case Feed page. Accept the defaults and click save.



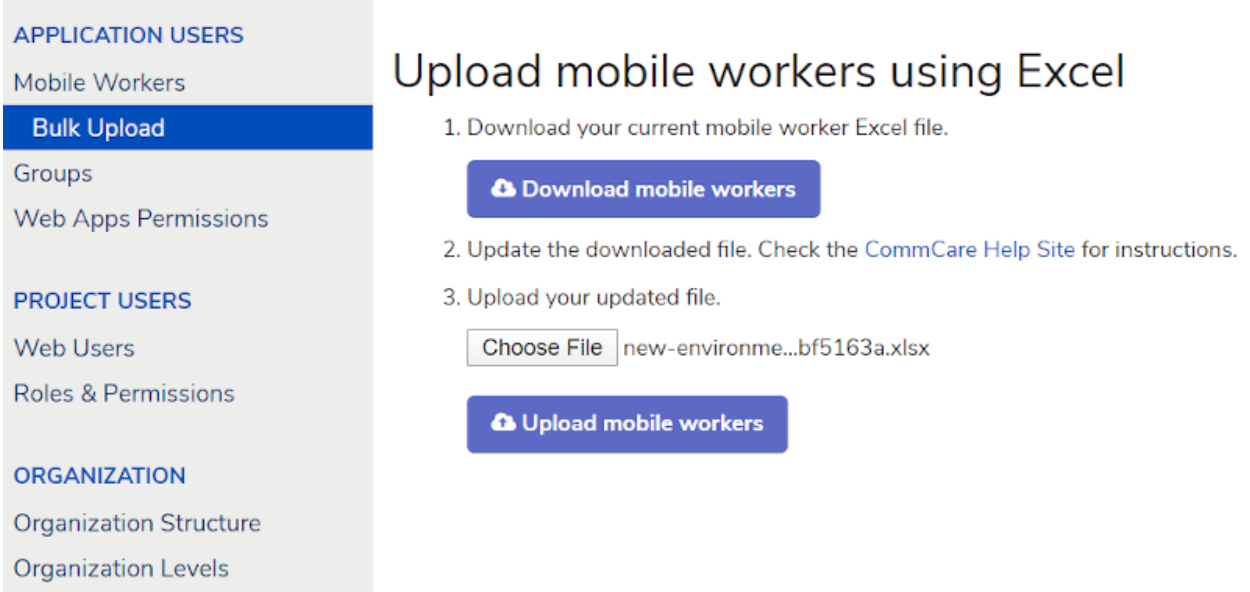
This will return you back to the PowerBi/Tableau Integration page with a newly created OData Feed. To test, we'll simply view the feed's data in the URL line of your browser, rather than connect to a BI tool. Click 'Copy OData Feed Link' and then open a new tab in your browser. Paste in this link and enter your CommCare HQ credentials. In a couple moments, you'll see your case information in JSON format in your browser

This step tests: At least one API endpoint within CommCare. This routine also uses Elasticsearch.

3.4.12 Step 7: Mobile worker upload

Lastly, let's validate a data upload in CommCare HQ. Navigate back to Users in the top ribbon and click Mobile Workers. Next, click 'Download Mobile Workers' and once the file is ready, 'Download Users.' This will download an xlsx file with Mobile Worker information. Upon opening the file; you'll see your mobile worker's info with its password hidden. Add additional usernames and passwords and save.

Once complete, click 'Return to manage mobile workers' and then the 'Bulk upload' button. Choose your file and once selected, click 'Upload mobile workers.'



After a moment, you will see 'Mobile Worker upload has finished - Successfully uploaded X mobile workers.'

This step tests: This tests Celery and Redis.

3.4.13 Step 8: (Optional) SMS Gateway functionality

If your project is going to use SMS capability or wants to explore that option in the future, we recommend testing this step as well.

To begin, access Messaging from the top ribbon and select View All. Here, you'll need to set up a new gateway by following the instructions in this article.

Gateway	Description	Supported Countries	Status	Actions
MOBILE_BACKEND_TWILIO	Twillio is another US-based phone number. International messages will be more expensive. The phone number for sending/receiving messages is +1 617 300 0456.	Multiple*	DEFAULT GATEWAY	Remove As Default

To test, return to Users and Mobile Workers. Then, select your mobile worker from the list to access the Edit Mobile Worker page. This page will display all information associated with the mobile worker and allow us to add a phone number. Scroll down to 'Add a Phone Number' and enter a phone number you're able to access and are comfortable using to receive a couple text messages. Please enter the phone number number, including country code, in digits only. Once done, select 'Add Number.'

Remaining on the Edit Mobile Worker page, click the 'Verify' button. Within a few moments, you should receive a text message. Replying '123' will allow your number to be verified and will grant that user full access to SMS features in CommCare.

This step tests: The connection of your SMS Gateway.

3.5 Migrating CommCare HQ

This section describes how to migrate CommCare HQ data between different instances.

- *Migrate a Project from one instance to a new instance* describes how to migrate a single project from one CommCare HQ instance to a new CommCare HQ instance.
- *Migrating an entire CommCare HQ instance* describes how to migrate an entire CommCare HQ instance from one set of servers to another set of servers.

3.5.1 Migrate a Project from one instance to a new instance

This document describes the process of migrating an individual project from Dimagi's cloud environment (or any other environment) to a new environment. If you are looking to migrate the entire environment to the new environment please see *Migrating an entire CommCare HQ instance*.

This process requires assistance from Dimagi if migrating from www.commcarehq.org, as some steps require administrative access to the old environment. To do that, reach out to Dimagi or file a support request.

There are two components to migrate an individual project.

1. Migrating the project specific data to the new environment. This is done using export and import data functionalities in CommCare HQ.
2. Switching the mobile devices with CommCare apps to use the new environment. This is done using a interim proxy URL for mobile devices.

Note that during the export/import data phase the access to the project has to be disabled for mobile and web users, which might take considerable amount of time. This should be planned and communicated in advance for a smooth switchover to the new environment.

1. Switch mobile devices to a proxy URL

Each application maintains a series of URLs pointing to CommCare HQ environment used for various requests made by the mobile devices. Migrating to a new web address requires updating these URLs in all the mobile devices at the time of switching the environments after the data is migrated. Since rolling out an app update to every mobile device of the project takes time during which the site needs to be offline, it will result in a long downtime for the project. Hence, if your project has any more than a couple of devices, it's best not to do it this way.

Instead, before the migration itself, a new/proxy URL can be set up and configured to direct requests to the original environment and the mobile devices can be gradually updated to use the new URL while the project is still online. Then after the migration, the URL can be switched to the new environment. The URL switch happens at the DNS level, so an app update is not needed. Note that, an all device update is still required in this method, but the advantage is that it can be done before the migration.

Note: Mobile devices should be switched to the proxy URL well in advance of doing the data migration so as to make sure all mobile users updated their applications!

To do this, follow the below steps.

1. Set up a domain name to be used for the migration. Have it point to the old environment.
2. Add that domain name to the old environment's public.yml

```
ALTERNATE_HOSTS:
  - commcare.example.com
```

3. Update the list of valid hosts in nginx and Django, then restart services for it to take effect. After this, CommCare HQ should be accessible at the new domain name.

```
$ cchq <env> ansible-playbook deploy_proxy.yml
$ cchq <env> update-config
$ cchq <env> service commcare restart
```

4. Set up SSL certificate for the domain name.
5. Enable the feature flag `CUSTOM_APP_BASE_URL` for the project. This will need to be done by a site administrator.
6. For each app in the project, navigate to Settings > Advanced Settings, and enter in the domain name you created above.
7. Make a new build and test it out to ensure form submissions and syncs still work as usual.
8. Release the build and roll it out to all devices. You can refer to Application Status Report to make sure that all the mobile devices are using this build.

If you don't want to use the final domain to point to old environment, a different domain can also be used during migration. That is, there are three registered domain names, which can be called "old", "new", and "mobile". This table describes which domain name each type of user will access at each stage of the migration:

	web users	mobile workers
current state	access old domain	access old domain
pre-migration	access old domain	access mobile domain as alias for old
during downtime	access blocked	access mobile domain, but blocked
post-migration	access new domain	access mobile domain as alias for new
after clean-up	access new domain	access new domain directly

Only after all the devices are updated to use a new/mobile URL, you can proceed to the next step.

2. Pull the domain data from the old environment

The migration will require you to block data access to prevent loss of data created during the migration. If you would like to do a practice run, you will still need to block data access to ensure the exported data is in a clean state, and the data will need to be cleared before the real run.

During the downtime, mobile users will still be able to collect data, but they will be unable to submit forms or sync with the server.

- Block data access by turning on the `DATA_MIGRATION` feature flag (via HQ Web UI).
- Print information about the numbers in the database for later reference. This will take a while (15 mins) even on small domains. Tip: add `--csv` to the command to save the output in a csv file.
  - `./manage.py print_domain_stats <domain_name>`

- A site administrator will need to run the data dump commands. First run `$ df -h` to ensure the machine has the disk space to store the output. Then run the data dumps.
  - `./manage.py dump_domain_data <domain_name>`
  - `./manage.py run_blob_export --all <domain_name>`

Note: It is important to have the commit hash that `dump_domain_data` and `run_blob_export` were run from. If Dimagi does not provide you with this commit hash, please followup to ensure you are able to reference this hash in future steps.

- Transfer these files to the new environment.

Note: If you are not able to use your own domain for a test run and would like dump data for a test domain for practising or testing, please contact support with the subject "Request for test domain dump data for migration testing" and mention this page. We will provide you the above data for a test domain from our staging environment.

---

3. Prepare the new environment to be populated

- Ensure you are running the following steps from a release created using the CommCare version/commit hash that you should have been provided in Step 1. This ensures the database will be migrated to the same state it was in when the data was dumped.
- Setup a new environment by following *Deploy CommCare HQ*
- Follow steps in *How To Rebuild a CommCare HQ environment* to ensure your environment is in a clean state before attempting to import data.
- Proceed to step 4.

4. Import the data to the new environment

- Ensure you are running the following steps from a release created using the CommCare version/commit hash that you should have been provided in Step 1. This ensures the database will be migrated to the same state it was in when the data was dumped.
- Import the dump files (each blob file will need to be imported individually)
  - `./manage.py load_domain_data <filename.zip>`
  - `./manage.py run_blob_import <filename.tar.gz>`
- Rebuild elasticsearch indices
  - Rebuild the indices with the new data `./manage.py ptop_preindex --reset`
- Print the database numbers and compare them to the values obtained previously
  - `./manage.py print_domain_stats <domain_name>`
- Rebuild user configrable reports by running.
  - `./manage.py rebuild_tables_by_domain <domain_name> --initiated-by=<your-name>`
- Bring the site back up `$ commcare-cloud <env> downtime end`
- Enable domain access by turning off the `DATA_MIGRATION` feature flag on the new environment (via HQ Web UI).

5. Ensure the new environment is fully functional. Test all critical workflows at this stage.

- Check reports and exports for forms and cases migrated from the old environment.
- Download the application with a test user and submit some forms.
- Ensure that those new form submissions appear in reports and exports.
- Make a change to the application and ensure that it can be built.

6. Turn on the new environment

- If desired, configure rate limiting to throttle the backlog of pending form submissions to handle a dramatic spike in load.
- Change the DNS entry for the proxy URL to point to the new environment. This will cause mobile devices to contact the new servers, bringing them back on-line.
- The new site should now be ready for use. Instruct web users to access the new URL.
- The old domain should remain disabled for a while to avoid confusion.

7. Clean up

- Switch mobile devices to the new environment's URL. Reverse the steps taken previously, since the custom URL is no longer necessary.
- Once the success of the migration is assured, request that a site administrator delete the project space on the old environment.

Troubleshooting

When transferring data for very large projects, you may run into infrastructural issues with the dump and load process. This is somewhat unsurprising when you consider that you're dealing with the project's entire lifetime of data in a single pass. It may be helpful to break down the process into smaller pieces to minimize the impact of any failures.

Blob data is already separated from everything else, which is advantageous, given that it's likely to be the most voluminous source of data. The rest of the data comes from four "dumpers" - `domain`, `toggles`, `couch`, and `sql`. You may use `dump_domain_data`'s `--dumper` arg to run any one (or multiple) of these independently. Each dumper also deals with a number of models, which you can also filter. Before getting started, you should run `print_domain_stats` to get an idea of where the project has data (even though it's not comprehensive).

`domain` and `toggles` are trivially small. Assuming the project is on the SQL backend for forms and cases, the `couch` dumper is also *likely* to be several orders of magnitude smaller than `sql`. Possible exceptions to this are projects with very large numbers of users, gigantic fixtures, or those which use data forwarding, as they'll have a large number of `RepeatRecord`s. If any of these models reach into the six figures or higher, you might want to dump them in isolation using `--include`, then `--exclude` them from the "everything else" couch dump. If you don't care about a particular model (eg: old repeat records), they can simply be excluded.

```
$ ./manage.py dump_domain_data --dumper=couch --include=RepeatRecord <domain>
$ ./manage.py dump_domain_data --dumper=domain --dumper=toggles --dumper=couch --
↪exclude=RepeatRecord <domain>
```

Dumping `sql` data is a bit trickier, as it's relational, meaning for example that `SQLLocation` and `LocationType` must be dumped together, lest they violate the DB's constraint checking on import. Fortunately, as of this writing, the biggest models are in relative isolation. There are two form submission models and six case models, but they don't reference

each other or anything else. You should validate that this is still the case before proceeding, however. Here are some example dumps which separate out forms and cases.

```
$ ./manage.py dump_domain_data --dumper=sql --include=XFormInstanceSQL --
↪include=XFormOperationSQL <domain>
$ ./manage.py dump_domain_data --dumper=sql --include=CommCareCaseSQL --
↪include=CommCareCaseIndexSQL --include=CaseAttachmentSQL --include=CaseTransaction --
↪include=LedgerValue --include=LedgerTransaction <domain>
$ ./manage.py dump_domain_data --dumper=sql --exclude=XFormInstanceSQL --
↪exclude=XFormOperationSQL --exclude=CommCareCaseSQL --exclude=CommCareCaseIndexSQL --
↪exclude=CaseAttachmentSQL --exclude=CaseTransaction --exclude=LedgerValue --
↪exclude=LedgerTransaction <domain>
```

You may also want to separate out `BlobMeta` or `sms` models, depending on the project.

If the data was already split into multiple dump files, then you can just load them each individually. If not, or if you'd like to split it apart further, you'll need to filter the `load_domain_data` command as well. Each dump file is a zip archive containing a file for each dumper, plus a `meta.json` file describing the contents. This can be useful for deciding how to approach an unwieldly import. You can also specify which loaders to use with the `--loader` argument (`domain`, `toggles`, `couch`, `sql`). You can also provide a regular expression to filter models via the `--object-filter` argument. Refer to the `meta.json` for options.

Here are some useful examples:

```
# Import only Django users:
$ ./manage.py load_domain_data path/to/dump.zip --object-filter=auth.User

# Import a series of modules' models
$ ./manage.py load_domain_data path/to/dump.zip --object-filter='\b(?:data_
↪dictionary|app_manager|case_importer|motech|translations)'

# Exclude a specific model
$ ./manage.py load_domain_data path/to/dump.zip --object-filter='^((?!RepeatRecord).)*$'
```

Lastly, it's very helpful to know how long commands take. They run with a progress bar that should give an estimated time remaining, but I find it also helpful to wrap commands with the unix `date` command:

```
$ date; ./manage.py <dump/load command>; date
```

3.5.2 Migrating an entire CommCare HQ instance

This document describes high level steps to migrate an entire CommCare HQ instance from one set of servers to another set of servers. If you are looking to migrate just a single project to a new environment please see *Migrate a Project from one instance to a new instance*. You can also use that method if you care about only a single project in your environment and don't need other projects.

There are database and application services in a CommCare HQ instance. Only the data for database services need to be migrated. There is no data to be migrated for stateless application services such as Django/Nginx/Celery/Pillow. It is recommended that you understand the below high level steps, document necessary commands to be run prior and plan out a migration timeline since it involves downtime. The downtime amount depends on how large your data size is.

1. Setup the new environment. Naming the environment with a new name and the servers with new host names is helpful down the line.
2. Disable automated tasks on new cluster (cron, monit, restarter scripts).

3. Start the downtime by stopping nginx and django services on old environment. You can use `commcare-cloud service` command to do this.
4. Let the pillow/celery changes finish processing. This can be monitored using the monitoring dashboards.
5. Stop all the services and ensure that the databases are not getting any reads/writes.
6. Copy static site content from old to new nginx server. The data is in `/etc/nginx/.htpasswd*` and `/var/www/html/`.
7. **Migrate data from old machines to corresponding machines in the new environment.**
    - Setup necessary SSH permissions to transfer data between old and new machines.
    - For Postgres/Elasticsearch/CouchDB, you can rsync the data directory for between corresponding nodes in old and new environments.
    - You may want to use `commcare-cloud copy-files` command if you have large number of nodes.
    - Alternatively, data can be migrated using backups taken after the services have stopped.
    - For BlobDB, follow the migration steps depending on what software you are using to host BlobDB.
    - Update pl_proxy config using `./manage.py configure_pl_proxy_cluster`.
    - Setup postgres standby nodes if you are using. You can use setup_pg_standby playbook to do this.
8. Deploy a code deploy on new environment.
9. Reset kafka checkpoints (for pillows) by doing *KafkaCheckpoint.objects.all().update(offset=0)* in a django management shell.
10. Perform a test to make sure the data in old and new environments match. - You can use *print_domain_stats* management command on both envs to compare. - Export forms or cases with few filters and compare.
11. Perform functionality tests using *Testing your new CommCare Environment* using a test domain.
12. Perform basic sync/submission tests on a CommCare mobile device. Since a DNS name is not setup yet, you might have to use */etc/hosts* or proxy forwarding to point the mobile device to the new environment when using the web address. Or you can create a mobile app using *CUSTOM_APP_BASE_URL* pointing to the public IP of new environment.
13. If everything is good, you can flip the DNS of your website to the public address of the new environment.
14. Your monitoring dashboards should start registering requests coming to the new environment.

3.6 Go Live Checklist

There are many things to be set up correctly before your CommCare HQ instance is made accessible over the internet. This checklist helps you to plan a new environment setup and make sure that you have everything ready to go live.

1. Procure necessary compute, storage and any other hardware.
2. Procure licenses or necessary subscriptions if you will be using any external services such as Datadog, Sentry and Email/SMS gateways.
3. Install necessary virtualization software and Ubuntu operation system.
4. Secure the servers where CommCare HQ will be installed.
5. Make sure *Ports Required for CommCare HQ* are not blocked by any firewall or networking rules.
6. Install CommCare HQ using *Quick Install on Single Server* or *Install Using Commcare-Cloud on one or more machines*.

7. Make sure SSL is set up and do an application deploy. Both of these are described in the installation docs.
8. Test that your environment is set up correctly by *Testing your new CommCare Environment*.
9. If you are migrating follow *Migrate a Project from one instance to a new instance* or *Migrating an entire CommCare HQ instance* and make sure the migration is successful using the tests described in those guides.
10. Make sure you have backups and monitoring set up correctly.
11. Configure DNS for your host address to your nginx server's public IP to go live.

CHAPTER

FOUR

OPERATIONS AND MAINTENANCE

This section has details on how to perform various operations and maintain a healthy CommCare HQ instance.

- *Managing The Deployment* describes how to restart services, how to deploy CommCare HQ code changes and other advanced topics.
- *Monitoring and Alerting* describes how to use Datadog for monitoring your instance and what all monitoring metrics are important to keep your instance up and running always.
- *Set up Sentry for error logs* describes how to set up Sentry to track error logs generated by various services in your CommCare HQ instance.
- *Expectations for Ongoing Maintenance* gives guidelines on the minimum required maintenance to keep your instance upto date.
- Please refer to the Table of Contents below for more topics.

4.1 Managing The Deployment

This section describes how to restart services, how to deploy CommCare HQ code changes and other advanced topics.

Table of Contents

- *Server Management Basics*
  - *Manage services*
  - *Stop all CommCare HQ services*
  - *Handling a reboot*
  - *Update CommCare HQ local settings*
  - *Run Django Management Commands*
  - *A note about system users*
- *Deploying CommCare HQ code changes*
  - *Prerequisites*
  - *Step 1: Update* `commcare-cloud`
  - *Step 2: Deploy new CommCare HQ code to all machines*
  - *Step 3: Checking services once deploy is complete*
- *Advanced*

        - *Run a pre-index*
        - *Resume failed deploy*
        - *Roll back a failed deploy*
        - *Deploy static settings files*
        - *Deploying Formplayer*
        - *Formplayer static settings*
    - *Scheduling Deploys*
        - *CommCare deploy*
        - *Formplayer deploy*
        - *Local Settings deploy*
    - *Resolving problems with deploys*
        - *Local Settings Mismatch*

4.1.1 Server Management Basics

Manage services

To manage services you can use the `service` command

```
$ commcare-cloud <env> service postgresql [status|start|stop|restart]
$ commcare-cloud <env> service --help
```

Stop all CommCare HQ services

```
$ commcare-cloud <env> service commcare stop
$ commcare-cloud <env> service commcare start
```

OR

```
$ commcare-cloud <env> downtime start
$ commcare-cloud <env> downtime end
```

In addition to stopping all services this command will check to see if any processes are still running and give you the option of terminating them or waiting for them to stop.

Handling a reboot

When a server reboots there are a number of tasks that should be run to ensure that the encrypted drive is decrypted and all systems are brought back up.

```
$ commcare-cloud <env> after-reboot --limit <inventory name or IP> all
```

Update CommCare HQ local settings

To roll out changes to the `localsettings.py` file for Django or the `application.properties` file for Formplayer:

```
$ commcare-cloud <env> update-config
```

Note that you will need to restart the services in order for the changes to be picked up by the processes.

Run Django Management Commands

To run Django management commands we need to log into a machine which has Django configured. Usually we run these commands on the `django_manage` machine which is a `webworker` machine.

```
$ cchq <env> ssh django_manage
$ sudo -iu cchq   #Switch to cchq user
$ cd /home/cchq/www/<env>/current # Change directory to current django release folder
$ source python_env/bin/activate  # Activate the python virtual env
$ ./manage.py <command>  # Run the command
```

There is also an alternate method for running management commands which can be useful in certain situations:

```
$ cchq <env> django-manage <command> <options>
```

Here are some common examples:

```
# get a Django shell
$ cchq <env> django-manage shell

# get a DB shell
$ cchq <env> django-manage dbshell --database <dbalias>

# check services
$ cchq <env> django-manage check-services
```

A note about system users

`commcare-cloud` creates and uses the `ansible` user on machines that it manages. You should not login as this user or use it for other things other than automated tasks run by the ansible process. This is especially applicable when you have a `control` machine that runs other commcare processes.

4.1.2 Deploying CommCare HQ code changes

This document will walk you through the process of updating the CommCare HQ code on your server using `commcare-cloud`.

Prerequisites

Ensure that you have a working version of `commcare-cloud` which is configured to act on your monolith or fleet of servers. You can find more information on setting up `commcare-cloud` in *CommCare Cloud Reference*.

If you have followed installation/quick-monolith-install: `commcare-cloud` will be installed on the CommCare HQ server itself.

All commands listed here will be run from your control machine which has `commcare-cloud` installed.

Step 1: Update `commcare-cloud`

We first want to pull the latest code for `commcare-cloud` to make sure it has the latest bugfixes by running:

```
$ update-code
```

This command will update the `commcare-cloud` command from GitHub and apply any updates required. You can see exactly what this command does in this file.

Step 2: Deploy new CommCare HQ code to all machines

CommCare HQ is deployed using ansible , which ensures only the necessary code is deployed to each machine.

Envoke the `deploy` command by running:

```
$ commcare-cloud <env> deploy
```

where you will substitute <env> for the name of the environment you wish to deploy to.

Preindex Command

The first step in deploy is what we call a `preindex`, which updates any CouchDB views and Elasticsearch indices. This only runs when changes need to be made, and may take a while depending on the volume of data that you have in these data stores. You may need to wait for this process to complete in order to complete deploy.

If your server has email capabilities, you can look out for an email notification with the subject: `[<env>_preindex] HQAdmin preindex_everything may or may not be complete`. This will be sent to the `SERVER_EMAIL` email address defined in the Django settings file.

You can also try running:

```
$ commcare-cloud <env> django-manage preindex_everything --check
```

If this command exits with no output, there is still a preindex ongoing.

Step 3: Checking services once deploy is complete

Once deploy has completed successfully, the script will automatically restart each service, as required. You can check that the system is in a good state by running:

```
$ commcare-cloud <env> django-manage check_services
```

This will provide a list of all services which are running in an unexpected state.

You may also wish to monitor the following pages, which provide similar information if you are logged in to CommCare HQ as a superuser:

- `https://<commcare url>/hq/admin/system/`
- `https://<commcare url>/hq/admin/system/check_services`

4.1.3 Advanced

The following commands may be useful in certain circumstances.

Run a pre-index

When there are changes that require a reindex of some database indexes it is possible to do that indexing prior to the deploy so that the deploy goes more smoothly.

Examples of change that woud result in a reindex are changes to a CouchDB view, or changes to an Elasticsearch index.

To perform a pre-index:

```
$ commcare-cloud <env> preindex-views
```

Resume failed deploy

If something goes wrong and the deploy fails part way through you may be able to resume it as follows:

```
$ commcare-cloud <env> deploy --resume
```

Roll back a failed deploy

You may also wish to revert to a previous version of the CommCare HQ code if the version you just deployed was not working for some reason. Before reverting, you should ensure that there were no database migrations that were run during the previous deploy that would break if you revert to a previous version.

```
$ commcare-cloud <env> deploy commcare --resume=PREVIOUS_RELEASE
```

Deploy static settings files

When changes are made to the static configuration files (like `localsettings.py`), you will need to deploy those static changes independently.

```
$ cchq <env> update-config
```

Deploying Formplayer

In addition to the regular deploy, you must also separately deploy the service that backs Web Apps and App Preview, called formplayer. Since it is updated less frequently, we recommend deploying formplayer changes less frequently as well. Doing so causes about 1 minute of service interruption to Web Apps and App Preview, but keeps these services up to date.

```
commcare-cloud <env> deploy formplayer
```

Formplayer static settings

Some Formplayer updates will require deploying the application settings files. You can limit the local settings deploy to only Formplayer machines to roll these out

```
$ cchq <env> update-config --limit formplayer
```

4.1.4 Scheduling Deploys

CommCare deploy

For locally hosted deployments, we recommend deploying **once a week** (for example, every Wednesday), to keep up to date with new features and security patches.

Since CommCare HQ is an Open Source project, you can see all the new features that were recently merged by looking at the merged pull requests on GitHub.

Formplayer deploy

In addition to the regular deploy, we recommend deploying formplayer **once a month**.

Local Settings deploy

Settings generally only need to be deployed when static files are updated against your specific environment.

Sometimes changes are made to the system which require new settings to be deployed before code can be rolled out. In these cases, the detailed steps are provided in the changelog. Announcements are made to the Developer Forum in a dedicated category when these actions are needed. We strongly recommend that anyone maintaining a CommCare Cloud instance subscribe to that feed.

4.1.5 Resolving problems with deploys

This document outlines how to recover from issues which are enountered when performing deploys from `commcare-cloud`.

Make sure you are up to date with the above documented process for deploying code changes.

All commands listed here will be run from your control machine which has `commcare-cloud` installed.

Local Settings Mismatch

If local settings files don't match the state expected by ansible, the deploy will fail.

Potential Causes

If `commcare-cloud` is not up to date when a deploy is run, the resulting deploy may change the local configuration of services in unintended ways, like reverting localsettings files pushed from an up-to-date deploy. If `commcare-cloud` is then updated and a new deploy occurs, the deploy can fail due to the ambiguous state.

Example Error

Here is an example of this error which could result from

- User A updates `commcare-cloud` to add `newfile.properties` to `formplayer` and deploys that change
- User B deploys `formplayer` with an out-of-date `commcare-cloud` instance which doesn't include User A's changes
- User B updates `commcare-cloud` and attempts to deploy again

```
TASK [formplayer : Copy formplayer config files from current release]␣
↪*********************************************************************************************************
failed: [10.200.9.53] (item={u'filename': u'newfile.properties'}) => {"ansible_loop_var
↪": "item", "changed": false, "item": {"filename": "newfile.properties"}, "msg":
↪"Source /home/cchq/www/production/formplayer_build/current/newfile.properties not found
↪"}
```

Resolution

After updating `commcare-cloud` and ensuring everything is up to date, running a static settings deploy on the relevant machines should fix this problem, and allow the next deploy to proceed as normal.

4.2 Monitoring and Alerting

This is a guide on how to setup monitoring for your CommCare HQ instance for realtime visibility into what's happening in your instance and how to setup alerts.

Real time monitoring is essential to get insights into how the system is performing at a given moment and troubleshoot issues as they arise. Monitoring could also be helpful to forecast resources requirements for future scaling. Alerts can be setup on various monitoring metrics to detect resource limits, anamolies that might cause an issue on your server.

CommCare HQ instance can send metrics to Datadog or Prometheus. This can be configured via commcare-cloud.

4.2.1 Datadog

Datadog is a monotoring and alerting tool that has support for variety of applications and is easily extendable which is why in our case it is used for monitoring various system, application metrics and also custom CommCare HQ metrics. You can read more about datadog in their docs

commcare-cloud can setup the requisite datadog integration automatically when you do full stack deploy. You will need to set `DATADOG_ENABLED` to True in your environment's public.yml file and add your account api keys to your vault file.

The default configuration sets up all datadog integrations and there might be a lot of data metrics being generated in your datadog account by your CommCare instance. Individual integrations may be turned on/off by setting relvant vars like datadog_integration_postgres or datadog_integration_redis etc in public.yml file.

4.2.2 Prometheus

Prometheus is an open source tool which you can self host on one of your machines. To setup a Prometheus server using commcare-cloud you can add a server under `promethus` and do `cchq <env> aps deploy_prometheus.yml`. This also gets executed automatically while doing `cchq deploy-stack`. To setup service integrations with the Prometheus server you will need to set `prometheus_monitoring_enabled` to True in your environment's public.yml file.

4.2.3 CommCare Infrastructure Metrics

Below we list down important metrics that you should consider as a minimum monitoring setup. Using these metrics you can create various dashboard views and alerts inside your Datadog project. Even though, this is specific to Datadog, you can adapt the same to Prometheus.

Recommended Dashboards

Here are few non-exhaustive preset dashboard views that you can import using Datadog import dashboard through json functionality.

- `hq-vitals.json` gives a glance of all components of CommCare
- `mobile-success.json` for monitoring success rate of mobile requests to the server
- `postgres-overview.json` for Postgres monitoring
- `celery.json` for monitoring bakcground application tasks of celery
- `couchdb.json` for CouchDB monitoring

The tables below tabulate most of the metrics already found in the *.json* Datadog dashboards listed above, but in some instances contain some additional metrics you might also want to consider monitoring. The list is not absolute nor exhaustive, but it will provide you with a basis for monitoring the following aspects of your system:

- Performance
- Throughput
- Utilization
- Availability
- Errors
- Saturation

Each table has the following format:

Metric	Metric type	Why care	User impact	How to measure
Name of metric	*Category or aspect of system the metric speaks to*	*Brief description of why metric is important*	*Explains the impact on user if undesired reading is recorded*	*A note on how the metric might be obtained. Please note that it is assumed that Datadog will be used as a monitoring solution unless specified otherwise.*

General Host

The Datadog Agent ships with an integration which can be used to collect metrics from your base system. See the System Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
CPU usage (%)	Utilization	Monitoring server CPU usage helps you understand how much your CPU is being used, as a very high load might result in overall performance degradation.	Lagging experience	system.cpu.idle system.cpu.system system.cpu.iowait system.cpu.user
Load averages 1-5-15	Utilization	Load average (CPU demand) over 1 min, 5 min and 15 min which includes the sum of running and waiting threads. What is load average	User might experience trouble connecting to the server	system.load.1 system.load.5 system.load.15
Memory	Utilization	It shows the amount of memory used over time. Running out of memory may result in killed processes or more swap memory used, which will slow down your system. Consider optimizing processes or increasing resources.	Slow performance	system.mem.usable system.mem.total
Swap memory	Utilization	This metric shows the amount of swap memory used. Swap memory is slow, so if your system depends too much on swap, you should investigate why RAM usage is so high. Note that it is normal for systems to use a little swap memory even if RAM is available.	Server unresponsiveness	system.swap.free system.swap.used
Disk usage	Utilization	Disk usage is important to prevent data loss in the event that the disk runs out of available space.	Data loss	system.disk.in_use
Disk latency	Throughput	The average time for I/O requests issued to the device to be served. This includes the time spent by the requests in queue and the time spent servicing them. High disk latency will result in slow response	Slow performance	system.io.await

Gunicorn

The Datadog Agent ships with an integration which can be used to collect metrics. See the Gunicorn Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Requests per second	Throughput	This metric shows the rate of requests received. This can be used to give an indication of how busy the application is. If you're constantly getting a high request rate, keep an eye out for bottlenecks on your system.	Slow user experience or trouble accessing the site.	gunicorn.requests
Request duration	Throughput	Long request duration times can point to problems in your system / application.	Slow experience and timeouts	gunicorn.request.duration.*
Http status codes	Performanc	A high rate of error codes can either mean your application has faulty code or some part of your application infrastructure is down.	User might get errors on pages	gunicorn.request.status.*
Busy vs idle Gunicorn workers	Utilization	This metric can be used to give an indication of how busy the gunicorn workers are over time. If most of the workers are busy most of the time, it might be necessary to start thinking of increasing the number of workers before users start having trouble accessing your site.	Slow user experience or trouble accessing the site.	gunicorn.workers

Nginx

The Datadog Agent ships with an integration which can be used to collect metrics. See the Nginx Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Total requests	Throughput	This metric indicates the number of client requests your server handles. High rates means bigger load on the system.	Slow experience	nginx.requests.total
Requests per second	Throughput	This metric shows the rate of requests received. This can be used to give an indication of how busy the application is. If you're constantly getting a high request rate, keep an eye out for services that might need additional resources to perform optimally.	Slow user experience or trouble accessing the site.	nginx.net.request_per_s
Dropped connections	Errors	If NGINX starts to incrementally drop connections it usually indicates a resource constraint, such as NGINX's worker_connections limit has been reached. An investigation might be in order.	Users will not be able to access the site.	nginx.connections.dropped
Server error rate	Error	Your server error rate is equal to the number of 5xx errors divided by the total number of status codes. If your error rate starts to climb over time, investigation may be in order. If it spikes suddenly, urgent action may be required, as clients are likely to report errors to the end user.	User might get errors on pages	nginx.server_zone.responses.5xx nginx.server_zone.responses.total_coun
Request time	Performance	This is the time in seconds used to process the request. Long response times can point to problems in your system / application.	Slow experience and timeouts	Need to include in NGINX configuration file

PostgreSQL

PostgreSQL has a statistics collector subsystem that collects and reports on information about the server activity.

The Datadog Agent ships with an integration which can be used to collect metrics. See the PostgreSQL Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Sequential scans on table vs. Index scans on table	Other	This metric speaks directly to the speed of query execution. If the DB is making more sequential scans than indexed scans you can improve the DB's performance by creating an index.	Tasks that require data to be fetched from the DB will take a long time to execute.	***PostgreSQL*:** pg_stat_user_tables ***Datadog integration*:** postgresql.seq_scans, postgresql.index_scans
Rows fetched vs. returned by queries to DB	Throughput	This metric shows how effectively the DB is scanning through its data. If many more rows are constantly fetched vs returned, it means there's room for optimization.	Slow experience for tasks that access large parts of the database.	***PostgreSQL*:** pg_stat_database ***Datadog integration*:** postgresql.rows_fetched, postgresql.rows_returned
Amount of data written temporarily to disk to execute queries	Saturation	If the DB's temporary storage is constantly used up, you might need to increase it in order to optimize performance.	Slow experience for tasks that read data from the database.	***PostgreSQL*:** pg_stat_database ***Datadog integration*:** postgresql.temp_bytes
Rows inserted, updated, deleted (by database)	Throughput	This metric gives an indication of what type of write queries your DB serves most. If a high rate of updated or deleted queries persist, you may want to keep an eye out for increasing dead rows as this will degrade DB performance.	No direct impact	***PostgreSQL*:** pg_stat_database ***Datadog integration*:** postgresql.rows_inserted, postgresql.rows_updated, postgresql.rows_deleted
Locks	Other	A high lock rate in the DB is an indication that queries could be long-running and that future queries might start to time out.	Slow experience for tasks that read data from the database.	***PostgreSQL*:** pg_locks ***Datadog integration*:** postgresql.locks
Deadlocks	Other	The aim is to have no deadlocks as it's resource intensive for the DB to check for them. Having many deadlocks calls for reevaluating execution logic. Read more	Slow experience for tasks that read data from the database. Some tasks may even hang and the user will get errors on pages.	***PostgreSQL*:** pg_stat_database ***Datadog integration*:** postgresql.deadlocks

Elasticsearch

The Datadog Agent ships with an integration which can be used to collect metrics. See the Elasticsearch Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Query load	Utilization	Monitoring the number of queries currently in progress can give you a rough idea of how many requests your cluster is dealing with at any particular moment in time.	A high load might slow down any tasks that involve searching users, groups, forms, cases, apps etc.	elasticsearch.primaries.search.query.current
Average query latency	Throughput	If this metric shows the query latency is increasing it means your queries are becoming slower, meaning either bottlenecks or inefficient queries.	Slow user experience when generating or reports, filtering groups or users, etc.	elasticsearch.primaries.search.query.tota elasticsearch.primaries.search.query.tim
Average fetch latency	Throughput	This should typically take less time than the query phase. If this metric is constantly increasing it could indicate problems with slow disks or requesting of too many results.	Slow user experience when generating or reports, filtering groups or users, etc.	elasticsearch.primaries.search.fetch.total elasticsearch.primaries.search.fetch.time
Average index latency	Throughput	If you notice an increasing latency it means you may be trying to index too many documents simultaneously.Increasing latency may slow down user experience.	Slow user experience when generating or reports, filtering groups or users, etc.	elasticsearch.indexing.index.total elasticsearch.indexing.index.time
Average flush latency	Throughput	Data is only persisted on disk after a flush. If this metric increases with time it may indicate a problem with a slow disk. If this problem escalates it may prevent you from being able to add new information to your index.	Slow user experience when generating or reports, filtering groups or users, etc. In the worst case there may be some data loss.	elasticsearch.primaries.flush.total elasticsearch.primaries.flush.total.time
Percent of JVM heap currently in use	Utilization	Garbage collections should initiate around 75% of heap use. When this value is consistently going above 75% it indicates that the rate of garbage collection is not	Users might experience errors on some pages	jvm.mem.heap_in_use

CouchDB

The Datadog Agent ships with an integration which can be used to collect metrics. See the CouchDB Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Open databases	Availability	If the number of open databases are too low you might have database requests starting to pile up.	Slow user experience if the requests start to pile up high.	couchdb.couchdb.open_databas
File descriptors	Utilization	If this number reaches the max number of available file descriptors, no new connections can be opened until older ones have closed.	The user might get errors on some pages.	couchdb.couchdb.open_os_files over
Data size	Utilization	This indicates the relative size of your data. Keep an eye on this as it grows to make sure your system has enough disk space to support it.	Data loss	couchdb.by_db.file_size
HTTP Request Rate	Throu put	Gives an indication of how many requests are being served.	Slow performance	couchdb.couchdb.httpd.requests
Request with status code of 2xx	Performanc	Statuses in the 2xx range are generally indications of successful operation.	No negative impact	couchdb.couchdb.httpd_status_c
Request with status code of 4xx and 5xx	Performanc	Statuses in the 4xx and 5xx ranges generally tell you something is wrong, so you want this number as low as possible, preferably zero. However, if you constantly see requests yielding these statuses, it might be worth looking into the matter.	Users might get errors on some pages.	couchdb.couchdb.httpd_status_c
Workload - Reads & Writes	Performanc	These numbers will depend on the application, but having this metric gives an indication of how busy the database generally is. In the case of a high workload, consider ramping up the resources.	Slow performance	couchdb.couchdb.database_read
Average request latency	Throu put	If the average request latency is rising it means somewhere exists a bottleneck that needs to be addressed.	Slow performance	couchdb.couchdb.request_time.a
Cache hits	Other	CouchDB stores a fair amount of user credentials in memory to speed up the authentication process. Monitoring usage of the authentication cache can alert you for possible attempts to gain unauthorized access.	A low number of hits might mean slower performance	couchdb.couchdb.auth_cache_hi
Cache misses	Error	If CouchDB reports a high number of cache misses, then either the cache is undersized to service the volume of legitimate user requests, or a brute force password/username attack is taking place.	Slow performance	couchdb.couchdb.auth_cache_m

Kafka

The Datadog Agent ships with a Kafka Integration to collect various Kafka metrics. Also see Integrating Datadog, Kafka and Zookeper.

Broker Metrics

Metric	Metric type	Why care	User impact	How to measure
UnderReplicated-Partitions	Availability	If a broker becomes unavailable, the value of UnderReplicatedPartitions will increase sharply. Since Kafka's high-availability guarantees cannot be met without replication, investigation is certainly warranted should this metric value exceed zero for extended time periods.	Fewer in-sync replicas means the reports might take longer to show the latest values.	kafka.replication.under_replicated_parti
IsrShrinksPerSec	Availability	The rate at which the in-sync replicas shrinks for a particular partition. This value should remain fairly static. You should investigate any flapping in the values of these metrics, and any increase in *IsrShrinksPerSec* without a corresponding increase in *IsrExpandsPerSec* shortly thereafter.	As the in-sync replicas become fewer, the reports might take longer to show the latest values.	kafka.replication.isr_shrinks.rate
IsrExpandsPerSec	Availability	The rate at which the in-sync replicas expands.	As the in-sync replicas become fewer, the reports might take longer to show the latest values.	kafka.replication.isr_expands.rate
TotalTimeMs	Performance	This metrics reports on the total time taken to service a request.	Longer servicing times mean data-updates take longer to propagate to the reports.	kafka.request.produce.time.avg kafka.request.consumer.time.avg kafka.request.fetch_follower.time.avg
ActiveController-Count	Error	The first node to boot in a Kafka cluster automatically becomes the controller, and there can be only one. You should alert on any other value that lasts for longer than one second. In the case that no controller is	Reports might not show new updated data, or even break.	kafka.replication.active_controller_coun

Producer Metrics

Metric	Metric type	Why care	User impact	How to measure
Request rate	Throughput	The request rate is the rate at which producers send data to brokers. Keeping an eye on peaks and drops is essential to ensure continuous service availability.	Reports might take longer to reflect the latest data.	kafka.producer.request_rate
Response rate	Throughput	Average number of responses received per second from the brokers after the producers sent the data to the brokers.	Reports might take longer to reflect the latest data.	kafka.producer.response_rat
Request latency average	Throughput	Average request latency (in ms). Read more	Reports might take longer to reflect the latest data.	kafka.producer.request_laten
Outgoing byte rate	Throughput	Monitoring producer network traffic will help to inform decisions on infrastructure changes, as well as to provide a window into the production rate of producers and identify sources of excessive traffic.	High network throughput might cause reports to take a longer time to reflect the latest data, as Kafka is under heavier load.	kafka.net.bytes_out.rate
Batch size average	Throughput	To use network resources more efficiently, Kafka producers attempt to group messages into batches before sending them. The producer will wait to accumulate an amount of data defined by the batch size. Read more	If the batch size average is too low, reports might take a longer time to reflect the latest data.	kafka.producer.batch_size_a

Consumer Metrics

Metric	Metric type	Why care	User impact	How to measure
Reco lag	Performanc	Number of messages consumers are behind producers on this partition. The significance of these metrics' values depends completely upon what your consumers are doing. If you have consumers that back up old messages to long-term storage, you would expect records lag to be significant. However, if your consumers are processing real-time data, consistently high lag values could be a sign of overloaded consumers, in which case both provisioning more consumers and splitting topics across more partitions could help increase throughput and reduce lag.	Reports might take longer to reflect the latest data.	kafka.consumer_lag
Reco consume rate	Throu put	Average number of records consumed per second for a specific topic or across all topics.	Reports might take longer to reflect the latest data.	kafka.consumer.records_
Fetch rate	Throu put	Number of fetch requests per second from the consumer.	requests per second from the consumer.	kafka.request.fetch_rate

Zookeeper

The Datadog Agent ships with an integration which can be used to collect metrics. See the Zookeeper Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Outstanding requests	Saturation	This shows the number of requests still to be processed. Tracking both outstanding requests and latency can give you a clearer picture of the causes behind degraded performance.	Reports might take longer to reflect the latest data.	zookeeper.outstanding_requests
Average latency	Throu put	This metric records the amount of time it takes to respond to a client request (in ms).	Reports might take longer to reflect the latest data.	zookeeper.latency.avg
Open file descriptors	Utilization	Linux has a limited number of file descriptors available, so it's important to keep an eye on this metric to ensure ZooKeeper can continue to function as expected.	Reports might not reflect new data, as ZooKeeper will be getting errors.	zookeeper.open_file_descriptor

Celery

The Datadog Agent ships with a HTTP Check integration to collect various network metrics. In addition, CommCare HQ reports on many custom metrics for Celery. It might be worth having a look at Datadog's Custom Metrics page. Celery also uses Celery Flower as a tool to monitor some tasks and workers.

Metric	Metric type	Why care	User impact	How to measure
Celery uptime	Availability	The uptime rating is a measure of service availability.	Background tasks will not execute (sending of emails, periodic reporting to external partners, report downloads, etc)	network.http.can_connect
Celery uptime by queue	Availability	The uptime rating as per queue.	Certain background or asynchronous tasks will not get executed. The user might not notice this immediately.	CommCare HQ custom metric
Time to start	Other	This metric shows the time (seconds) it takes a task in a specific queue to start executing. If a certain task consistently takes a long time to start, it might be worth looking into.	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	CommCare HQ custom metric
Blockage duration by queue	Throughput	This metric indicates the estimated time (seconds) a certain queue was blocked. It might be worth it to alert if this blockage lasts longer than a specified time.	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	CommCare HQ custom metric
Task execution rate	Throughput	This metric gives a rough estimation of the amount of tasks being executed within a certain time bracket. This can be an important metric as it will indicate when more and more tasks take longer to execute, in which case an investigation might be appropriate.	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	CommCare HQ custom metric
Celery tasks by host	Throughput	Indicates the running time (seconds) for celery tasks by host.	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	CommCare HQ custom metric
Celery tasks by queue	Throughput	Indicates the running time (seconds) for celery tasks by queue. This way you can identify slower queues.	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	CommCare HQ custom metric
Celery tasks by task	Throughput	Indicates the running time (seconds) for celery tasks by each respective task. Slower tasks can be identified.	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	CommCare HQ custom metric
Tasks queued	Saturation	Indicates the number of tasks queued by each respective queue. If this becomes increasingly large, keep an	For the most part this might go unnoticed for the user, but there	Celery

RabbitMQ

The Datadog Agent ships with an integration which can be used to collect metrics. See the RabbitMQ Integration for more information.

Metric	Metric type	Why care	User impact	How to measure
Queue depth	Saturation	Using queue depth, messages ready and messages unacknowledged	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	rabbitmq.queue.messages
Messages ready	Other	Using queue depth, messages ready and messages unacknowledged	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	rabbitmq.queue.messages_read
Messages unacknowledged	Error	Using queue depth, messages ready and messages unacknowledged	Certain background tasks will fail to execute, like sending emails, SMS's, alerts, etc.	rabbitmq.queue.messages_una
Queue memory	Utilization	RabbitMQ keeps messages in memory for faster access, but if queues handle a lot of messages you could consider using lazy queues in order to preserve memory. Read more	For the most part this might go unnoticed for the user, but there will be a delay in the execution of background tasks, like sending emails, SMS's, alerts, etc.	rabbitmq.queue.memory
Queue consumer	Other	The number of consumers is configurable, so a lower-than-expected number of consumers could indicate failures in your application.	Certain background tasks might fail to execute, like sending emails, SMS's, alerts, etc.	rabbitmq.queue.consumers
Node sockets	Utilization	As you increase the number of connections to your RabbitMQ server, RabbitMQ uses a greater number of file descriptors and network sockets. Since RabbitMQ will block new connections for nodes that have reached their file descriptor limit, monitoring the available number of file descriptors helps you keep your system running.	Background tasks might take longer to execute as, or in the worst case, might not execute at all.	rabbitmq.node.sockets_used
Node file descriptors	Utilization	As you increase the number of connections to your RabbitMQ server, RabbitMQ uses a greater number of file descriptors and network sockets. Since RabbitMQ will block new connections for nodes that have reached their file descriptor limit, monitoring the available number of file descriptors helps you keep your system running.	Background tasks might take longer to execute as, or in the worst case, might not execute at all.	rabbitmq.node.fd_used

4.3 Set up Sentry for error logs

sentry.io is an error logging platform that CommCare HQ server is integrated with. This is a guide on how setup this integration. If you would like to self host Sentry using commcare-cloud, see sentry-on-prem.

4.3.1 Register account on sentry.io

Go to sentry.io to create your account.

As of Dec 2018, there's a free tier that allows you to log up to 5k errors a month.

Sentry allows you to divide your account into multiple "projects". To log formplayer errors as well, we recommend creating a separate projects called "commcarehq" and "formplayer". If you'd rather not, it should be possible to send both errors to the same project.

4.3.2 Configure for your account

Each account and project on Sentry will come with its own set of IDs and keys, which you'll have to store in the environment configuration failes. A complete configuration looks like this:

public.yml

```
localsettings:
  SENTRY_ORGANIZATION_SLUG: 'organization slug'
  SENTRY_PROJECT_SLUG: 'commcare project slug'
  # Repository name for integrating commits into releases
  SENTRY_REPOSITORY: 'dimagi/commcare-hq'
```

vault.yml

```
localsettings_private:
  SENTRY_DSN: 'https://{key}@sentry.io/{project_id}'
  # This token is used to create releases and deploys. It needs the 'project:releases'␣
↪permission.
  SENTRY_API_KEY: '{token}'

  ...
secrets:
  FORMPLAYER_SENTRY_DSN: 'https://{key}@sentry.io/{project_id}'
  ...
```

For more details see the Sentry docs.

4.4 Expectations for Ongoing Maintenance

Like any software project, CommCare HQ requires active maintenance to be kept up-to-date. As project maintainers, Dimagi frequently makes changes such as library upgrades, bug fixes, feature development, database migrations, and infrastructure improvements. Whenever possible, these changes are made to be rolled out automatically with a regular deploy, but occasional larger changes require direct handling to minimize disruptions.

To minimize friction, we recommend that anyone hosting an instance of CommCare HQ commit to keeping their environment up to date by following the guidelines described on this page. It can become much more challenging to update an environment that has been neglected for an extended period.

4.4.1 Monitor the developers forum

Subscribe to the developers forum to keep in contact with Dimagi and other parties hosting CommCare. Dimagi will announce important changes there, such as upcoming upgrades.

4.4.2 Deploy CommCare HQ at least once every two weeks

CommCare HQ is under continuous development, so to ensure you are running an up-to-date version of the code, you should be deploy changes at least every two weeks.

Some code changes are meant to be rolled out over the course of two or three deploys, which allows us to minimize or eliminate the disruptions caused by things like backwards-incompatible database changes. It is important for the developers to be able to make assumptions about how these changes will impact existing environments.

4.4.3 Update commcare-cloud before every deploy and check the changelog

`commcare-cloud` is developed in conjunction with CommCare HQ. To be on the safe side, it's best to update just before deploying CommCare HQ. When you do so, check for new entries in the changelog, which should alert you of any new changes which might require you to take action. We recommend you take a look through the current entries to get an idea of what these changes might look like.

Take the action described in each new changelog entry within the compatibility window, if applicable. Aside from urgent security issues, there should be a window during which you can plan for downtime or for a more involved upgrade. If you will be unable to apply the change in the window, please reach out on the forum.

After the window expires, Dimagi will drop support for the old version. You may then face additional difficulty in deploying the change, as well as incompatibility problems on your server.

4.5 Supporting Your Users

The following guide explains how to set up processes/systems in order to provide support to various users who may interact with your CommCare HQ instance such as mobile users, application building teams and reporting teams etc.

4.5.1 Why set up a Support System

CommCare is a powerful technology that caters to programs of different nature from simple data collection from a few hundred users to large scale deployment of millions of users. As with any technology, users using CommCare will need support beyond the user documentation to be successful at their job. To this extent, setting up a robust support system becomes essential for the success of any project in which CommCare is set.

Identifying stakeholders

Projects using CommCare often have stakeholders who use and interact with CommCare HQ in different ways and they often have varied levels of technical expertise. Some examples of the stakeholders are community health / frontline workers (CHWs / FLWs), supervisors who monitor the FLWs, Monitoring and Evaluation (M&E) staff and data teams that deal with the data analytics and reporting side of the program, project team that is managing the CommCare HQ project setup and high level program staff. These users interact with CommCare mobile and server in different ways and will need different types of support.

Below are some example support issues that might come up from various stakeholders of the program

- An FLW not being able to sync or submit data to CommCare HQ server
- A project team member not seeing upto data on system

When setting up a support system we recommend you to do an exercise of identifying the stakeholders in your program and think through the program/technical workflows that these stakeholders would be part of and the potential issues that they might encounter.

4.5.2 Components of a Support System

A good support system should have the following components:

- Training and documentation with common support questions and issues for end users
- Support channels for users to get the support they need
- Support team members
- Support processes and tools to track, triage, troubleshoot and escalate incoming issues

Below we give details on each component.

Training and Documentation

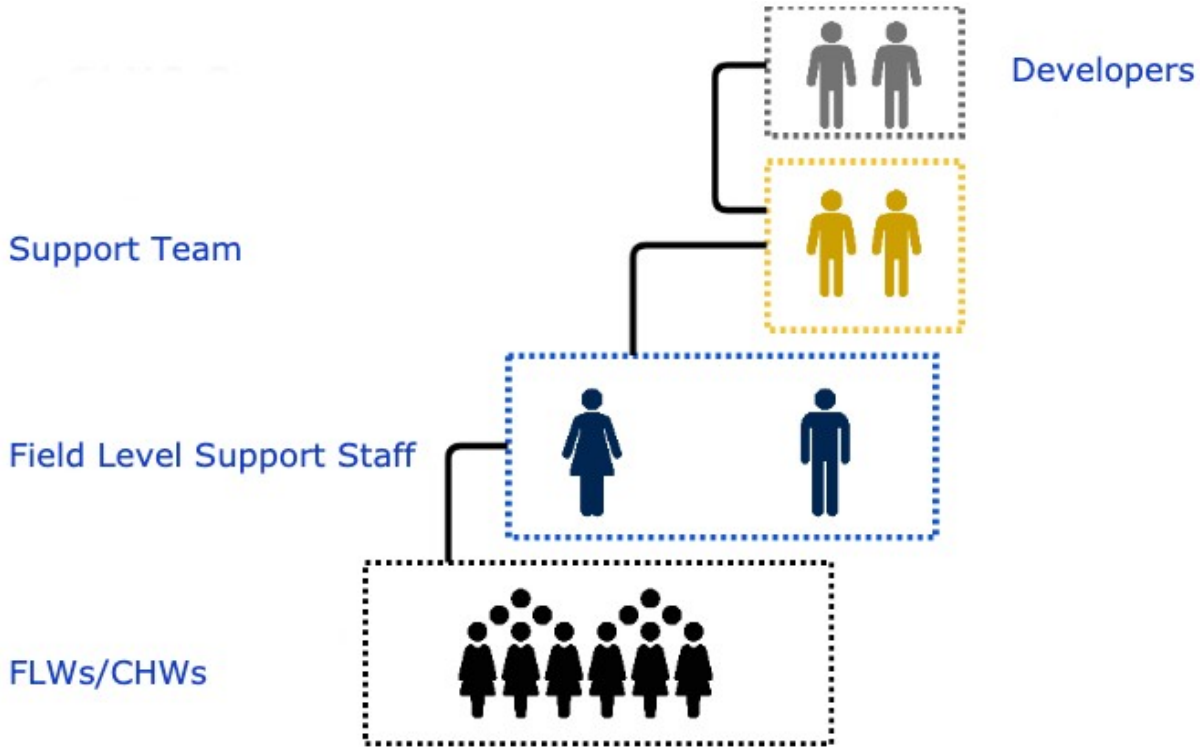
A major portion of support issues end up being simply queries from users who may not know how to perform an action. The best way to avoid these is by providing training at the time of onboarding the users and documentation on how to use the system.

The documentation should also contain common queries that users might have. The support process should allow for these docs to be continuously updated as new repeating support queries are raised.

Support Channels for users to get the support they need

Support channels are required to enable communication between the end users facing an issue and the support team who have the expertise to resolve the issues. There are various possible support channels depending on how the support process is set up.

- Report an Issue from CommCare app. For this,
  - An FLW/CHW should be trained to use 'Report an Issue' in the CommCare mobile.
  - When not able to use the app CHW/FLW should have an offline mechanism of reporting issues to supervisors or higher level program staff for support.
- 'Report an Issue' button on CommCare HQ Server
- Support Email: Higher level program staff should have access to a Direct support email address that auto creates tickets in a helpdesk app.
- An issue tracker CommCare application that has 'Raise an Issue' module which tracks the issues for all FLWs for supervisor's reference



These channels are suitable for different stakeholders of the program depending on the level of their expertise in being able to communicate online. For example, a frontline worker may not be able to communicate via email with support in which case a direct phone support with a supervisor can be more easy. On the other hand if there are too many FLWs under a supervisor where a phone support is not possible, an Issue Tracker application could be useful.

Support team

Support team could be a one or multi person team of experts who have a good overall understanding of the system and can troubleshoot and answer various issues coming from the users. The team should have direct access to developer team members who can troubleshoot issues that are beyond the scope of the support team's knowledge. The team will also need administrative level access to troubleshoot certain issues. The program and developer team should create policies to enable the support team without compromising on the security. The support team would also own the overall support process and improve it as necessary to achieve higher Service Level Agreements (SLAs).

Support Processes

The support process is a defined process of how incoming support issues are handled from receiving the issues up till the resolution of these issues.

In its simplest form the support process might involve a developer answering all the issues from all the users of the system either via email or via phone by keeping track of these issues simply in an email or list of notes. This could work when the program is very small. This process would break down when the number of users increase even slightly. For programs of anything more than 20 FLWs we recommend a proper support process to handle and resolve all incoming issues in a timely manner.

Depending on the scale and complexity of the program either a basic or advanced process would become necessary. Below we describe the two processes in detail.

Basic Support Process and Tools

At a minimum, we recommend the below setup for handling support.

- Set up a helpdesk app such as Jira, Zendesk or other open source helpdesk app. Or if you are already using a project management software, you could use that instead.
- Set up a dedicated support email where all support queries can be sent to either via directly or via Report an Issue button on CommCare HQ. Configure this in your server using support email param in your environment.
- Integrate the helpdesk software with the support email such that all the incoming emails create individual tickets in the helpdesk software.
- Helpdesk software should have below fields
  - Title and Description of the issue
  - Status: To describe the status of the ticket such as incoming, waiting for user's input, being worked on and resolved etc as you see fit
  - Assignee: This allows the ticket to be passed between various team members depending on their expertise.
  - Priority: This is a very important field. Please see below the section on priority
  - Any additional fields as you see fit for project management needs.
- Onboard various members of support, program and developer team members to the helpdesk app as necessary.

Priority field

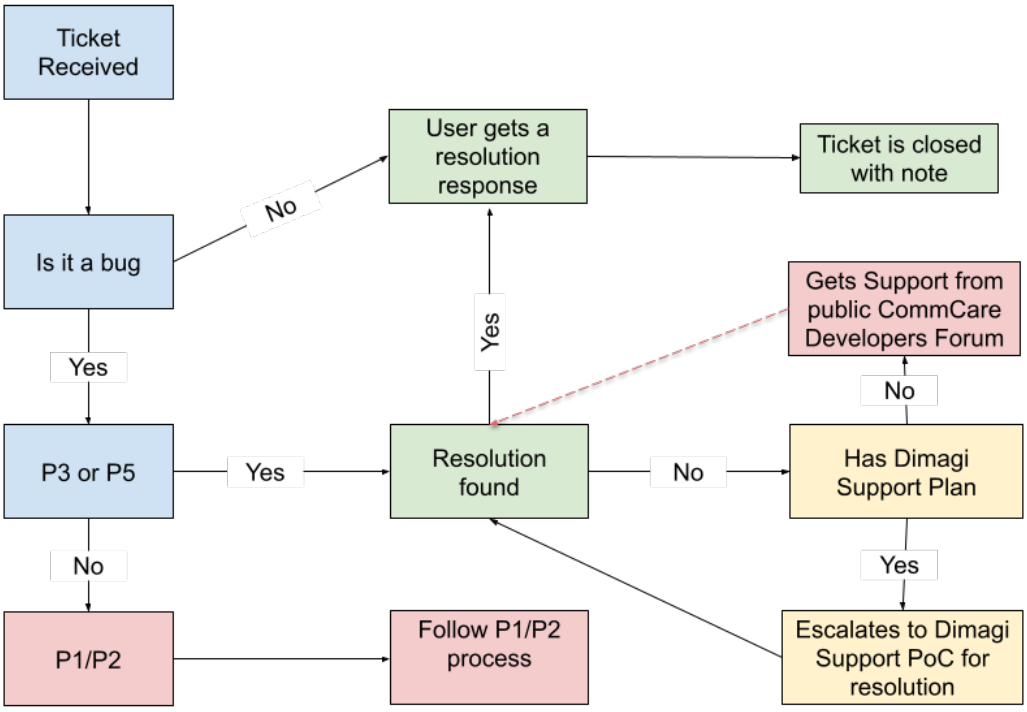
A priority level such as P1, P2, P3, P4 etc that describes the urgentness of the ticket and the number of users it's affecting. It's good to have a team-wide common definition on what each priority level means and document it in a relevant place for everyone's reference. Below is a suggested priority level based on Dimagi's support process.

- **P1** : Severe (a blocker), don't do anything else. May have to sleep less tonight. There is (business loss) already. The longer it's not fixed, the longer the product and the team are in failure state. Examples: Site down, data loss, security breakdown etc.
- **P2** : A problem which has made an important/critical function unusable or unavailable and no workaround exists. Examples: All users not being able to sync with server.
- **P3** : High (Should be fixed), if not fixed, will lose integrity in product. Example: Pillows falling behind by a large backlog.
- **P4** : Normal (Should be fixed, time and resources permitting)
- **P5** : Low

The priority level helps the entire support team and developers to understand how they should prioritize the particular ticket. A support team member triaging the ticket can setup the priority.

Ticket Workflow

Once the support system is set up below is a general process that can be followed. Note that for P1/P2 we recommend a separate on-call like process stated in the *P1/P2 Process* recommendations.



- An issue is reported view UI or directly
- A ticket is created in helpdesk app automatically or support creates it if the issue is reported via email/chat.
- When a new ticket arrives,
  - A support team member performs the initial investigation
  - If more information is required to resolve the issue the user is contacted for more information.
  - If the ticket fits P1/P2 criteria, follow P1/P2 process
  - Support team member updates the fields of the ticket such as priority, status and assignee.
  - Depending on the ticket, the support team member might resolve and respond back to the user or escalate it to a different team member from the program or developer team.
  - If the ticket priority is low, the team might put it into a backlog that can be reviewed later.
  - If the team is not able to get resolve, the ticket can be reported to Dimagi support directly if the team has a support plan or else to the public CommCare developers forum
- Once the resolution is found the support team member sends the resolution to the user and closes the ticket after updating relevant ticket fields.

Apart from this a regular periodical (weekly or biweekly) team calls could also be used to coordinate the overall support activities.

P1/P2 Process

The standard support process stated above works well for tickets with priority lower than P2. As defined above tickets with priority P1 indicate a very urgent ticket that affects all users, which may be causing a downtime or irreversible data loss/corruption or other critical issues. P2 priority indicates a critical function being available that might soon result in a P1 issue if neglected. Given that there is a lot of urgency tied to P1 and P2, we recommend a separate process to resolve these issues.

The intention of a separate P1/P2 process is to address below unique expectations associated with P1 or P2 incidents.

1. Fix the issue as soon as possible
2. Establish communication with users and stakeholders to inform about the issue
3. Followup Actions such as Root Cause Analysis to prevent issues like this from getting repeated

We recommend below a general process that addresses these three expectations. You may tweak it as you see fit in your organizational context or even create your own process but in the least it should address the above three expectations.

Process for P1/P2

1. Kick off the process
    a) Create a ticket and mark it's priority to P1
    b) Form and gather an Incident Response Team consisting of a Developer lead who is the main developer to resolve the issue, a Response manager who makes sure the developer has all the resources to resolve the issue other strategic planning around the issue and Support lead to handles communication with external users and internal teams
    c) Do a P1 call with Incident Response Team members to troubleshoot and co-ordinate next steps on the issue. Create a shared live P1 document to add notes on the issue.
    d) Response manager or support lead announces the issue in the internal and external channels to let various stakeholders be informed about the issue. Various mechanisms exist to facilitate this
        i) Dedicated internal/external chat groups
        ii) CommCare HQ Alerts Page (<yourhqserver.com>/alerts) has an alerts page where a new banner can be set up if the site is not down.
        iii) Tools such as statuspage.io
2. Manage the issue
    a) Response manager or support lead should periodically check in with the developer lead to understand the status and make sure the developer lead has all the support to resolve the issue in a timely manner.
    b) Post updates on the communication channels regarding the status and ETA.
3. After the issue is resolved
    a) Announce that the issue is resolved on various communication channels
    b) Take down any banners or update tools such as statuspage.io
    c) Change the priority of the ticket from P1 to other appropriate priority.
    d) Update the status of the ticket to 'Pending Retro'
4. Doing a Retrospective

a) Ask the developer lead to create a retrospective document that details the root cause of the issue and steps to be taken to prevent such issue from repeating in the future. The developer can use techniques such as Five Whys to do the retrospective.

b) Schedule a Retrospective meeting with a wider team to discuss the retrospective and do a postmortem analysis on the ticket to arrive at a comprehensive list of action items to prevent such issues from repeating and make process related improvements to minimize the resolution time.

The main difference between a P1 and P2 issue is the urgency with which the issue needs to be resolved. The same process is recommended for P2 issues with relaxations in urgency which means it may not need as frequent and close monitoring as P1.

Advanced Support Process and Tools

Programs that are very large scale could produce a very high volume of support tickets that need to be resolved under SLAs. This requires more advanced support systems to be setup at multiple levels of the program in an escalating manner. This often needs to be planned as a core facet of the program from the ground up. A support system at this level usually consists of

- Issue Tracker Applications to supervisors to support FLWs
- Helpdesks at District/Block level and escalation process
- Program level support team at the top
- View into SLAs

There is no general setup that can be recommended to all the projects as each program has different needs at scale.

4.5.3 Support System Implementation checklist

As discussed in *Components of a Support System*, in order to implement a good support system all of the necessary components need to be in place. You can use the below checklist to make sure you have a robust support system in place.

1. Make sure enough training material and documentation exists for end users to prevent support queries.
2. Establish support channels with various stakeholders
3. Create a support team
4. Create documentation that outlines
   a) Definitions of various priorities
   b) The support processes for regular and P1/P2 tickets.

CHAPTER

FIVE

HOW TO SCALE

This section contains guides on how to test your CommCare HQ instance's performance and how to estimate sizing for higher number of users using a tool called commcare_resource_model.

The hardware requirements required for instances under 15,000 mobile users is given at *Hardware requirements and Deployment Options*. This section is intended to provide information on how to scale CommCare HQ for projects going beyond 15,000 mobile users.

5.1 Performance Benchmarking for CommCare HQ using Locust

5.1.1 Introduction

Performance testing is useful to understand how well your CommCare HQ instance performs under a given load. The results of this testing will inform any bottlenecks in your setup and how to scale it for more number of users.

Dimagi uses Locust, and scripts stored in the commcare-perf repository, for performance benchmarking apps on CommCare HQ.

The commcare-perf repository includes instructions for installing the scripts.

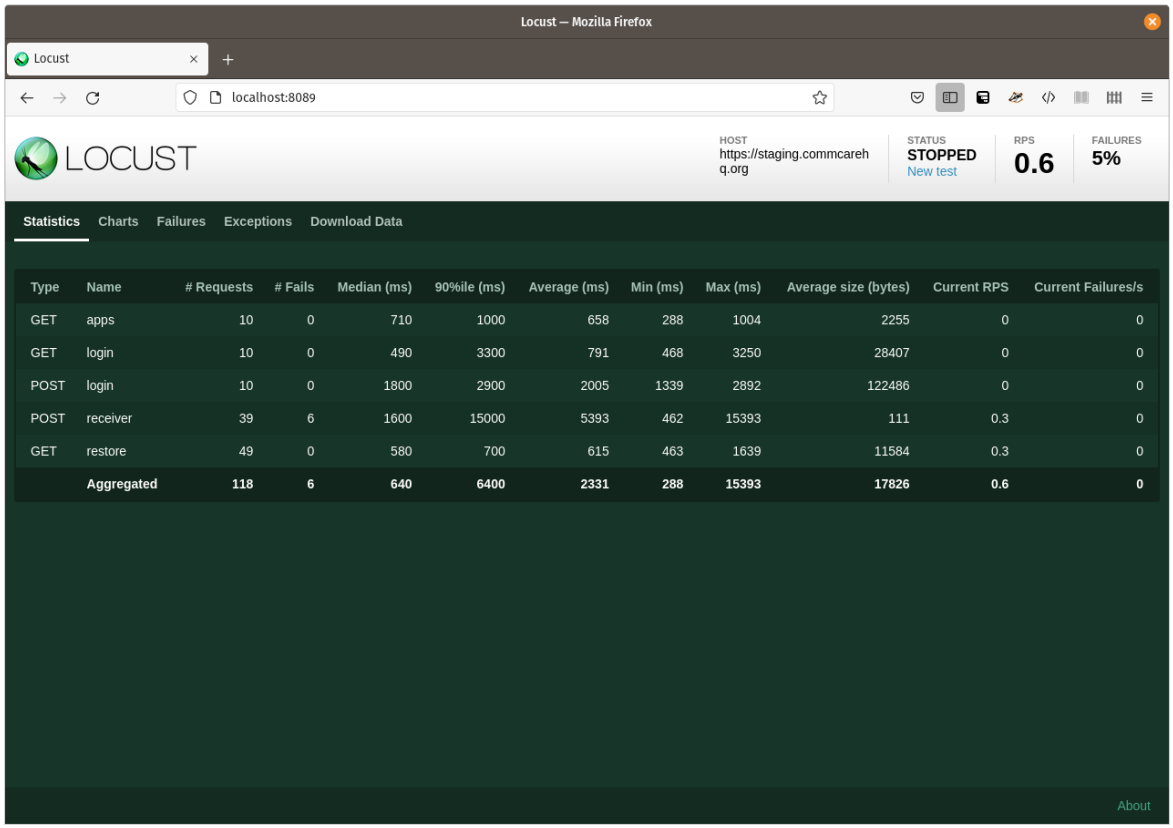
See Locust's documentation for running Locust in Docker.

5.1.2 Getting started

1. Start Locust with a web interface:

```
(venv) $ locust -f locustfiles/form_submission.py
```

2. Open http://localhost:8089 in a browser.
3. You will be asked the number of users to simulate, and their spawn rate. Start with 1 user, at a spawn rate of 1 user per second, and click the "Start swarming" button.
4. Locust will show you a table of statistics. You can switch to the "Charts" tab to see requests per second, response times, and the number of simulated users.



5. Click “Stop” and increase the number of users to see how CommCare is affected.

5.1.3 Submitting your own forms

`locustfiles/form_submission.py` uses three example form submissions for testing. They are saved in the `xforms/` directory.

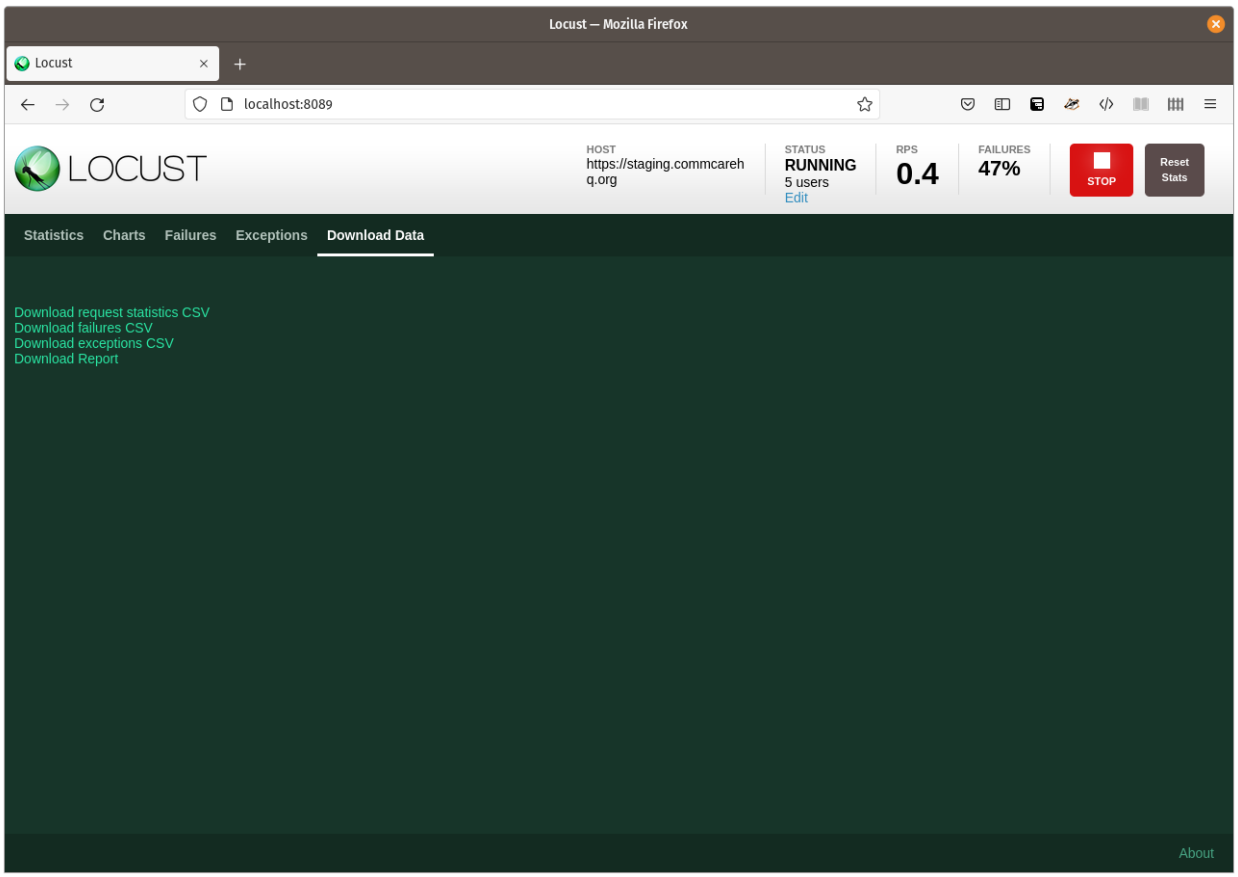
If you would like to use your own form submissions, you can find them in CommCare.

1. Navigate to “Reports” > “Case List”, and find a sample case with test data. (Do not use a real case.)
2. Select the case, and choose the “Case History” tab.
3. Select a form. (If you select the form that registered the case, you will get duplicate cases.)
4. Choose “Raw XML”.
5. Copy and paste the XML into a file and save it in the `xforms/` directory.
6. Repeat steps 1 to 5 for as many forms as you wish to submit.
7. `locustfiles/form_submission.py` will submit all the files it finds in the `xforms/` directory. If you don’t want to use the original examples, just delete them.
8. Restart Locust to test using your form submissions.

5.1.4 Saving results

In Locust, switch to the "Download Data" tab.

Here you can download the report, and various data in CSV format.



5.2 How to Estimate Infrastructure Sizing

CommCare HQ has a distributed architecture and each service can be scaled horizontally as more users are added to the project. Gradual scaling by adding resources as and when needed is the best way to scale a CommCare HQ project. This works well in cloud environments where resources can be procured instantaneously. But this doesn't work well when doing on-premise hosting where the infrastructure is bought one time exclusively for the project and there isn't a flexibility to gradually add resources. So, a way to estimate the infrastructure required for a given number of users becomes necessary.

Given that CommCare HQ is a complex distributed application, estimating infrastructure sizing for a given number of users is a complex task, and it's often more of an art than science as it requires keen observation of resource usage at different points of load for a given number of users.

5.2.1 commcare_resource_model

The infrastructure required to scale to a higher specific number of users can be obtained by examining the amount of resources (storage, RAM and CPU) that are getting used for a known current number of users. One most brute way to estimate the resource requirement would be to directly increase the hardware proportionate to the number of users. For e.g. if the system resources are all at their full capacity for 1000 users, to scale to 2000 users approximately a double amount of resources will be necessary.

But in practice, it will be more accurate to analyze the correlation between number of users and corresponding resource usage via load parameters such as form submissions, case creations, case updates, multimedia submissions and syncs etc that affect the resource usage more directly and then use these ratios to estimate for higher number of users. Such analysis could be done by examining the resource usage for a given time period.

Once these load parameters and the relation to resource usage ratios are obtained, calculations can be done using a simple excel calculator to proportionately estimate the resources required for a higher number of users, but it would be complicated to manage/modify such an excel calculator.

For this reason, we have developed a python based tool called commcare_resource_model. It takes a config file containing values for various user load and resource capacity parameters and ratios to estimate the resources required for higher user loads. Once the script is run for the given config, it will generate an excel file containing the resource estimates for a higher number of users.

One limitation of this tool or more correctly the limitation of the sizing methodology used in this tool is that the resource capacity must be known for each individual service. This is easy when each service is hosted on a different machine. When multiple services share a resource, it is not possible to estimate the resource capacity per each individual service. The tool can be still used for storage estimation as it is easy to examine the storage usage per each individual service as there are separate data directories for each service.

5.2.2 How to use commcare_resource_model

Note that commcare_resource_model can be used to estimate sizing only when a baseline usage and load data is available after CommCare HQ has been used for a given number of users. It can't be used for estimating sizing for a new project. To understand what infrastructure you need for a new project please refer to sizing-buckets.

The general process for using commcare_resource_model is below.

1. Install the commcare_resource_model python tool
2. Create a configuration file that specifies number of users to be scaled to, load and usage parameters from your existing environment
3. Run the script

The configuration file has three main sections that specifies the baseline usage and load of the system. Below are the three sections.

- **Usage section** Under this section all usage parameters such as number of users by date ranges and its correlation to load parameters (form submissions, case creations, case updates, multimedia submissions and syncs etc) can be listed. These numbers and ratios can be obtained by project_stats_report management command and also by examining the individual services, processes running one each VM.
- **Services section** for resource calculations for each service. This section can refer to one or more usage parameters from usage section which specifies the amount of usage that resource can handle.
- **Summary dates** The dates for which the resources need to be estimated.
- Additional parameters such as estimation_buffer, storage_display_unit etc are available.

Please refer to the docs for this tool to get an understanding of how to use this tool.

CHAPTER

SIX

COMMCARE HQ SERVICES GUIDES

CommCare HQ uses many services such as Django, PostgreSQL, BlobDB, Kafka and many more. This section has information how a service is used and how to perform various operations related to the service.

6.1 PostgreSQL

Table of Contents

- *PostgreSQL*
  - *Usage in CommCare*
  - *Configuration*

6.1.1 Adding a postgresql hot standby node

The PostgreSQL standby is a hot standby (accept reads operations only) of each production database. Each Database node should have standby node configured and deployed. This will require configuring in the environment inventory files to set variables as follows:

On primary node

- `hot_standy_server` (points to standby server)
- `postgresql_replication_slots` (list of replication slots)
  - replication slots should be formatted a list as follows:
    - CSV invenory: “[“”slot1””,””slot2””]”
    - INI inventory: [“slot1”,”slot2”]

On the standby node

- `hot_standby_master` (point to primary)
- `replication_slot` (which replication slot to use)
- Add node to `pg_standby` group

To deploy the standby nodes we'd first need to create the replication slots in the primary. We normally use ansible playbook to perform this

```
$ commcare-cloud ap deploy_postgresql.yml --limit <primary host>
```

Note:- In case if a restart is not desired then this command can be used.

```
$ commcare-cloud <env> run-shell-command <primary-node> -b --become-user=postgres "psql -
↪d <database name> -c  "'"'"SELECT * FROM pg_create_physical_replication_slot('<slot␣
↪name>')"'"'""
```

After that we can use the `setup_pg_standby.yml` playbook

```
$ cchq <env> ap setup_pg_standby.yml -e standby=[standby node]
```

6.1.2 Promoting a hot standby to master

1. In your inventory you have two postgresql servers defined:
   - `pg_database` with `postgresql_replication_slots = ["standby0"]`
   - `pg_standby` where `hot_standby_master = pg_database` and `replication_slot = "standby0"`
2. Begin downtime for your site:

   ```
   $ commcare-cloud <env> downtime start
   ```

3. Verify that the replication is up to date

   ```
   $ commcare-cloud <env> run-shell-command pg_database,pg_standby 'ps -ef | grep -E
   ↪"sender|receiver"'
       [ pg_database ] ps -ef | grep -E "sender|receiver"
       postgres 5295 4517 0 Jul24 ? 00:00:01 postgres: wal sender process rep 10.116.
   ↪175.107(49770) streaming 0/205B598
       [ pg_standby ] ps -ef | grep -E "sender|receiver"
       postgres 3821 3808 0 Jul24 ? 00:01:27 postgres: wal receiver process streaming␣
   ↪0/205B598
   ```

   Output shows that master and standby are up to date (both processing the same log).

4. Promote the standby

   ```
   $ commcare-cloud <env> ansible-paybook promote_pg_standby.yml -e standby=pg_standby
   ```

5. **In your inventory remove `hot_standby_master` and `replication_slot` variables from your standby node,**
   and move the node from the `pg_standby` group to the `postgresql` group.

6. **Update the host for the applicable database(s) in `postgresql.yml` and update your processes to point to the newly**
   promoted server:

```
$ commcare-cloud <env> update-config
```

7. If the standby you've promoted is one of the `form_processing` databases update the PL proxy cluster configuration:

```
$ commcare-cloud <env> django-manage --tmux configure_pl_proxy_cluster
```

8. **[Optional] If you have configured your standby and master nodes to use different parameters, or**
   you would like to create replication slots on the newly promoted standby update those configurations:

```
$ commcare-cloud <env> ap deploy_db.yml --limit pg_database,pg_standby
```

9. End downtime for your site:

```
$ commcare-cloud <env> downtime end
```

10. If you would like to have another standby for this newly promoted master, follow above instructions for adding a standby database.

6.1.3 Splitting a shard in postgresql

This document describes the process required to split a partitioned database from one PostgreSQL instance into itself and another. This migration will require downtime.

Assumptions

For the purposes of this document we'll assume that we have three database machines, *pg1* , *pg2* and *pg3*. *pg1* has one database and *pg2* and *pg3* has none:

pg1 databases:

- *commcarehq_p1* (with django alias *p1*)

pg2 and *pg3* is a newly deployed server in the *[postgresql]* group and we want to create a new *commcarehq_p2* on *pg2* and *commcarehq_p3* on *pg3* with half the data from *commcarehq_p1* on each.

pg1 is currently the only database containing sharded data. Half of the data should be moved to a new *pg2* and *pg3* servers

Current database configuration:

```
PARTITION_DATABASE_CONFIG = {
    'shards': {
        'p1': [0, 3],
    },
    'groups': {
        'main': ['default'],
        'proxy': ['proxy'],
        'form_processing': ['p1'],
    },
}
```

(continues on next page)

(continued from previous page)

```
DATABASES = {
    'proxy': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'commcarehq_proxy',
        'USER': 'commcarehq',
        'PASSWORD': 'commcarehq',
        'HOST': 'pg1',
        'PORT': '5432',
        'TEST': {
            'SERIALIZE': False,
        },
    },
    'p1': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'commcarehq_p1',
        'USER': 'commcarehq',
        'PASSWORD': 'commcarehq',
        'HOST': 'pg1',
        'PORT': '5432',
        'TEST': {
            'SERIALIZE': False,
        },
    },
}
```

At the end of this process shards 0 & 1 should be on *pg2* and shards 2 & 3 will be on *pg3*.

Important Notes

By default pglogical does not replicate any DDL commands. This means that any CommCare HQ migrations may not be applied to the target databases while logical replication is active. It is recommended to not deploy any changes during the time when splitting the database. More technical information can be found at https://github.com/2ndQuadrant/pglogical/blob/REL2_x_STABLE/docs/README.md#ddl

Process Overview

1. Ensure that *pg1* is set to use logical replication
2. Setup *pg2* and *pg3* as a new database
3. Setup logical replication from pg1 to pg2 and pg3
4. Promote *pg2* and *pg3* to a master node
5. Update the applications configuration to go to *pg2* and *pg3*

Process detail

1. Setup logical replication on *pg1*

In your inventory file set the following vars for *pg1*:

```
[pg1]
...
postgresql_wal_level = logical
postgresql_max_worker_processes = 8
postgresql_shared_preload_libraries = ["pglogical"]
```

Also ensure that your replication user has superuser privileges on all databases, in vault.yml:

```
POSTGRES_USERS:
  replication:
    username: 'foo'
    password: 'bar'
    role_attr_flags: 'LOGIN,REPLICATION,SUPERUSER'
```

In postgresql.yml:

```
postgresql_hba_entries:
  - contype: host
    users: foo
    netmask: 'pg2 ip address'
  - contype: host
    databases: replication
    users: foo
    netmask: 'pg2 ip address'
  - contype: host
    users: foo
    netmask: 'pg3 ip address'
  - contype: host
    databases: replication
    users: foo
    netmask: 'pg3 ip address'
```

Then deploy these settings to your databases:

```
commcare-cloud <env> ap deploy_db.yml --limit=pg1,pg2,pg3
```

2. Setup *pg2* and *pg3*

Setup *pg2* and *pg3* as you would another postgresql database in commcare-cloud.

In addition to normal setup, add the following to your `postgresql.yml` file:

```
dbs:
  logical:
    - name: commcarehq_p2
      host: pg2
      master_host: pg1
```

(continues on next page)

(continued from previous page)

```
      master_db_name: commcarehq_p1
      replication_set: [0, 1]
    - name: commcarehq_p3
      host: pg3
      master_host: pg1
      master_db_name: commcarehq_p1
      replication_set: [2, 3]
```

Deploy this change to your databases using:

```
commcare-cloud <env> ap setup_pg_logical_replication.yml
```

This will begin the replication process in the background which replicates one table at a time. To check the progress:

```
ANSIBLE_DISPLAY_SKIPPED_HOSTS=False commcare-cloud <env> ap setup_pg_logical_replication.
↪yml --tags=status

TASK [All subscriber status] *****************
ok: [pg2] => {
    "msg": [
        [
            {
                "show_subscription_status": "(sub_name,initializing,provider_name,\
↪"connection_string\",internal_pg_logical_name,{subscription},{all})"
            }
        ]
    ]
}
```

In the above output `initializing` means that the database is copying from the original to the new database. Once complete it will change to `replicating`

3. Stop all DB requests

Once the databases are fully replicated and you are ready to switch to the new databases, bring the site down.

Stop all CommCare processes

```
commcare-cloud <env> downtime start
```

Stop pgbouncer

```
commcare-cloud <env> service postgresql stop --only pgbouncer --limit pg1,pg2,pg3
```

Verify that the replication is up to date by ensuring `replay_location` and `sent_location` are the same for each database:

```
ANSIBLE_DISPLAY_SKIPPED_HOSTS=False commcare-cloud <env> ap setup_pg_logical_replication.
↪yml --tags=status --limit=pg1
ok: [100.71.184.26] => {
    "msg": [
        [
```

(continues on next page)

(continued from previous page)

```
            {
                "application_name": "commcarehq_p2_0_1_sub",
                "replay_location": "2058/4C93E6B0",
                "sent_location": "2058/4C93E6B0"
            }
        ],
        [
            {
                "application_name": "commcarehq_p3_2_3_sub",
                "replay_location": "2058/4C93E6B0",
                "sent_location": "2058/4C93E6B0"
            }
        ]
    ]
}
```

Synchronize the sequences:

```
ANSIBLE_DISPLAY_SKIPPED_HOSTS=False commcare-cloud <env> ap setup_pg_logical_replication.
↪yml --tags=synchronize_sequences --limit=pg1
```

4. Update configuration

Update ansible config

Update the *dbs* variable in the environment's *postgresql.yml* file to show that the *p2* database is now on *pg2*:

```
...
 dbs:
 ...
   form_processing:
     ...
     partitions:
-      p1:
-        shards: [0, 3]
-        host: pg1
+      p2:
+        shards: [0, 1]
+        host: pg2
+      p3:
+        shards: [2, 3]
+        host: pg3
       ...
```

Deploy changes

```
# update localsettings
commcare-cloud <env> update-config

# update PostgreSQL config on new PG node
commcare-cloud <env> ap deploy_db.yml --limit=pg2,pg3
```

(continues on next page)

(continued from previous page)

```
# update the pl_proxy cluster
commcare-cloud <env> django-manage --tmux configure_pl_proxy_cluster
```

To remove the logical replication run the following on all subscriber databases:

```
SELECT pglogical.drop_node(sub_name, true)
```

5. Restart services

start pgbouncer

```
commcare-cloud <env> service postgresql start --only pgbouncer --limit pg2,pg3
```

Restart services

```
commcare-cloud <env> downtime end
```

6.1.4 Moving a PostgreSQL sharded database

For large scale deployments of CommCare HQ the database tables that contain case and form (and related) data can be partitioned between multiple PostgreSQL databases.

To ease the burden of scaling it is common to create as many databases as necessary up front since splitting databases is much more difficult than moving databases. In this case a deployment of CommCare HQ may start with a single database server that contains all the databases. As the project scales up more database servers can be added and the databases can be moved to spread the load.

This document describes the process required to move a database from one PostgreSQL instance to another.

For the purposes of this document we'll assume that we have two database machines, *pg1* and *pg2*. *pg1* has two partitioned databases and *pg2* has none:

pg1 databases:

- *commcarehq_p1* (with django alias *p1*)
- *commcarehq_p2* (with django alias *p2*)

pg2 is a newly deployed server and we want to move *commcarehq_p2* onto *pg2*.

Assumptions

- *pg2* has been added to the Ansible inventory and included in the *[postgresql]* group
- *pg2* has had a full stack Ansible deploy
- *pg1* has a replication slot available

Process Overview

1. Setup *pg2* node as a standby node of *pg1*
2. Promote *pg2* to a master node
3. Update the configuration so that requests for *p2* go to *pg2* instead of *pg1*.

Process detail

1. Setup *pg2* as a standby node of *pg1*

This step does not require downtime and can be done at any stage prior to the downtime.

```
commcare-cloud <env> ansible-playbook setup_pg_standby.yml -e standby=pg2 -e hot_standby_
↪master=pg1 -e replication_slot=[replication slot name]
```

2. Stop all DB requests

This will bring the CommCare HQ site down.

Stop all CommCare processes

```
commcare-cloud <env> service commcare stop
```

You may have to wait for any long running celery tasks to complete. You can list any celery workers that are still running using the following commands:

```
commcare-cloud <env> django-manage show_celery_workers
commcare-cloud <env> run-shell-command celery "ps -ef | grep celery"
```

Stop pgbouncer To be completely certain that no data will be updating during the move you can also prevent connections from pgbouncer:

```
pg1 $ psql -p 6432 -U someuser pgbouncer

> PAUSE commcarehq_p1
```

3. Check document counts in the databases

This step is useful for verifying the move at the end.

```
commcare-cloud <env> django-manage print_approximate_doc_distribution --csv
```

4. Update configuration

Update ansible config

Update the *dbs* variable in the environment's *postgresql.yml* file to show that the *p2* database is now on *pg2*:

```
...
 dbs:
 ...
   form_processing:
     ...
     partitions:
       p1:
         shards: [0, 1]
         host: pg1
       p2:
         shards: [2, 3]
-        host: pg1
+        host: pg2
       ...
```

Deploy changes

```
# update localsettings
commcare-cloud <env> update-config

# update PostgreSQL config on new PG node
commcare-cloud <env> ap deploy_db.yml --limit=pg2
```

5. Verify config changes

```
commcare-cloud <env> django-manage print_approximate_doc_distribution --csv
```

This should show that the *p2* database is now on the *pg2* host.

6. Promote *pg2* to master

Verify that replication is up to date

```
commcare-cloud <env> run-shell-command pg1,pg2 'ps -ef | grep -E "sender|receiver"'

    [ pg1 ] ps -ef | grep -E "sender|receiver"
    postgres 5295 4517 0 Jul24 ? 00:00:01 postgres: wal sender process rep 10.116.175.
↪107(49770) streaming 0/205B598

    [ pg2 ] ps -ef | grep -E "sender|receiver"
    postgres 3821 3808 0 Jul24 ? 00:01:27 postgres: wal receiver process streaming 0/
↪205B598
```

Output shows that master and standby are up to date (both processing the same log).

Promote *pg2*

```
commcare-cloud <env> run-shell-command pg2 -b 'pg_ctlcluster <pg version> main promote'
```

7. Verify doc counts

Re-run command from step 5 to verify that the document counts are the same.

8. Update pl_proxy config

```
commcare-cloud <env> django-manage configure_pl_proxy_cluster
```

9. Restart services

Unpause pgbouncer

```
pg1 $ psql -p 6543 -U someuser pgbouncer

> RESUME commcarehq_p1
```

Restart services

```
commcare-cloud <env> service commcare start
```

10. Validate the setup

One way to check that things are working as you expect is to examine the connections to the databases.

```
SELECT client_addr, datname as database, count(*) AS connections FROM pg_stat_activity␣
↪GROUP BY client_addr, datname;
```

pg1 should only have connections to the *commcarehq_p1* database

```
  client_addr   | database    | connections
----------------+-------------+------------
 <client IP>    | commcarehq_p1 |   3
```

pg2 should only have connections to the *commcarehq_p2* database

```
  client_addr   | database    | connections
----------------+-------------+------------
 <client IP>    | commcarehq_p2 |   3
```

11. Cleanup

Delete duplicate databases

Once you're confident that everything is working correctly you can go back and delete the duplicate databases on *pg1* and *pg2*.

pg1

```
DROP DATABASE commcarehq_p2;
```

pg2

```
DROP DATABASE commcarehq_p1;
```

Drop replication slot

In order to prevent the WAL logs on *pg1* from piling up we need to delete the replication slot that was used by *pg2*:

```
commcare-cloud <env> run-shell-command p1 -b --become-user postgres 'psql -c "select pg_
↪drop_replication_slot('\'<slot name>\'');"'

# optionally re-create the slot
commcare-cloud <env> run-shell-command p1 -b --become-user postgres 'psql -c "select pg_
↪create_physical_replication_slot('\'<slot name>\'');"'
```

Update PostgreSQL config

```
commcare-cloud <env> ap deploy_db.yml --limit=postgresql
```

Other useful commands

Check which nodes are in recovery

```
commcare-cloud <env> run-shell-command postgresql -b --become-user postgres "psql -c
↪'select pg_is_in_recovery();'"
```

Check replication slot status

```
commcare-cloud <env> run-shell-command postgresql -b --become-user postgres "psql -c
↪'select * from pg_replication_slots;'"
```

6.1.5 Upgrading PostgreSQL

1. Ensure that you have a full backup of all PostgreSQL data before starting the upgrade process.
2. Review the PostgreSQL documentation for the pg_upgrade tool.
3. Test the upgrade process locally with Vagrant or on a test environment.

In the below instructions:

```
HOSTS-TO-UPGRADE: list of hosts / host groups to upgrade include standbys
OLD-VERSION: Current version of PostgreSQL that is running
```

(continues on next page)

(continued from previous page)

```
NEW-VERSION: Version being upgraded to
OLD-PORT: Port that PostgreSQL is currently running on (defaults to 5432)
```

Upgrade preparation

1. Run the `deploy_postgres.yml` playbook to ensure your system is up to date

```
commcare-cloud <env> ap deploy_postgres.yml --limit HOSTS-TO-UPGRADE
```

2. Update PostgreSQL version and port

postgresql.yml

```
postgres_override:
    postgresql_version: NEW-VERSION
    postgresql_port: NEW-PORT  # this must be different from the current PostgreSQL port
```

3. Run the `deploy_postgres.yml` playbook to install the new version of PostgreSQL

```
commcare-cloud <env> ap deploy_postgres.yml --limit HOSTS-TO-UPGRADE --tags postgresql
```

Perform the upgrade

1. Stop all processes connecting to the databases

```
commcare-cloud <env> run-module HOSTS-TO-UPGRADE service "name=pgbouncer state=stopped"
```

2. Run the upgrade playbooks

The following commands can be used to perform the upgrade:

```
commcare-cloud <env> ansible-playbook pg_upgrade_start.yml --limit HOSTS-TO-UPGRADE \
    -e old_version=OLD-VERSION -e new_version=NEW-VERSION [-e old_port=OLD-PORT]

commcare-cloud <env> ansible-playbook pg_upgrade_standbys.yml --limit HOSTS-TO-UPGRADE \
    -e old_version=OLD-VERSION -e new_version=NEW-VERSION [-e old_port=OLD-PORT]

commcare-cloud <env> ansible-playbook pg_upgrade_finalize.yml --limit HOSTS-TO-UPGRADE \
    -e old_version=OLD-VERSION -e new_version=NEW-VERSION
```

Follow the instructions given in the play output.

3. Revert to using the old port

Once you are satisfied with the upgrade you can revert the port change in `postgresql.yml` and apply the changes.

```
commcare-cloud <env> ansible-playbook deploy_postgres.yml --limit HOSTS-TO-UPGRADE
```

4. Upgrade the psql client

Once the postgres upgrade is completed, update the psql client on all hosts

```
commcare-cloud <env> ansible-playbook deploy_common.yml
```

6.1.6 Migrate plproxy to new node

This Migration doesn't require downtime since the databases do not contain any data.

1. Update the new node in env config:
   - Replace the old nodes in `<env>/postgresql.yml` with new nodes
2. Run `cchq <env> deploy-stack --limit=<new nodes>` on new nodes.
3. Run `cchq <env> update-config --limit=django_manage`
   - Expect to see changes to django settings for this server
4. Validate on `django_manage` machine:
   - Log into `django_manage machine`, switch to `cchq user`, navigate to `/home/cchq/<env>/current/` and activate python virtual environment
   - Run `python manage.py configure_pl_proxy_cluster --create_only`
   - Run `env CCHQ_IS_FRESH_INSTALL=1 python manage.py migrate --database <Django DB alias Name>` (proxy,proxy_standby)
   - Validate settings
     - `python manage.py check_services`
     - `python manage.py check --deploy -t database`
5. Run `cchq <env> update-config` (no limit)
6. Restart mobile webworkers
   - `cchq <env> service commcare restart --limit=mobile_webworkers`
7. Check for errors
   - Load the site in a browser
   - Watch monitoring dashboards for errors
   - Watch Sentry for errors
8. Restart remaining services
   - `cchq <env> service commcare restart --limit= 'all:!mobile_webworkers'`
9. Check for errors again (see step 7)
10. Cleanup:

- Remove old plproxy nodes from env config (inventory etc)

6.1.7 Upgrading CitusDB

1. Ensure that you have a full backup of all PostgreSQL data before starting the upgrade process.
2. Review the documentation:
   - pg_upgrade
   - citus
3. Test the upgrade process locally with Vagrant or on a test environment.

This upgrade is split into two parts:

1. Upgrade the 'citus' extension
2. Upgrade PostgreSQL

The citus extension should be upgraded prior to upgrading PostgreSQL.

In the below instructions:

```
OLD-VERSION: Current version of PostgreSQL that is running
NEW-VERSION: Version being upgraded to
OLD-PORT: Port that PostgreSQL is currently running on (defaults to 5432)
```

Upgrade 'citus'

Prepare for the 'citus' extension upgrade

1. Run the `deploy_citusdb.yml` playbook to ensure your system is up to date

```
commcare-cloud <env> ap deploy_citusdb.yml
```

Upgrade the 'citus' extension

1. Update `public.yml`

public.yml:

```
citus_version: <new citus version>
```

2. Run the `deploy_citusdb.yml` playbook

```
commcare-cloud <env> ansible-playbook deploy_citusdb.yml --tags citusdb -e postgresql_
↪version=OLD-VERSION
```

3. Check the extension version

```
commcare-cloud <env> run-shell-command citusdb -b --become-user postgres "psql -d␣
↪DATABASE -c '\dx' | grep citus"
```

Upgrade PostgreSQL

Prepare for the PostgreSQL upgrade

1. Update `public.yml`

public.yml:

```
citus_postgresql_version: NEW-VERSION
citus_postgresql_port: NEW-PORT  # this must be different from the current port
```

2. Run the `deploy_citusdb.yml` playbook to install the new version of PostgreSQL

```
commcare-cloud <env> ansible-playbook deploy_citusdb.yml --tags citusdb
```

Perform the upgrade

1. Stop all processes connecting to the databases

```
commcare-cloud <env> run-module citusdb_master -b service "name=pgbouncer state=stopped"
```

2. Run the upgrade playbooks

The following commands can be used to perform the upgrade:

```
commcare-cloud <env> ansible-playbook pg_upgrade_start.yml --limit citusdb \
    -e old_version=OLD-VERSION -e new_version=NEW-VERSION [-e old_port=OLD-PORT]

commcare-cloud <env> ansible-playbook pg_upgrade_standbys.yml --limit citusdb \
    -e old_version=OLD-VERSION -e new_version=NEW-VERSION [-e old_port=OLD-PORT]

commcare-cloud <env> ansible-playbook pg_upgrade_finalize.yml --limit citusdb \
    -e old_version=OLD-VERSION -e new_version=NEW-VERSION
```

Follow the instructions given in the play output.

3. Revert to using the old port

Once you are satisfied with the upgrade you can revert the port change in `public.yml` and apply the changes.

```
commcare-cloud <env> ansible-playbook deploy_citusdb.yml
```

6.1.8 Usage in CommCare

PostgreSQL is one of the primary databases that CommCare uses.

There are a few different categories of data that are stored in PostgreSQL:

- System level metadata:
  - billing
  - notifications
  - etc
- Project level metadata:
  - users
  - locations
  - etc
- Project generated data:
  - forms
  - cases
  - ledgers
  - message logs
  - formplayer sessions
  - syclogs
- Project reporting data
  - custom report tables
  - standard report tables

CommCare is configured to work with a number of databases:

- `commcarehq`
  - the main 'metadata' database that stores users, locations etc.
- `formplayer`
  - used only by formplayer to store user sessions etc.
- `commcarehq_ucr`
  - custom report database
- `commcarehq_synclogs`
  - data necessary for maintaining the state of the mobile devices
- `commcarehq_proxy`

  - routing database for sharded databases
  - this is only required for installations that require data sharding due to the scale
- `commcarehq_p[N]`
  - database partition storing cases, forms, ledgers and messaging data
  - there can be any number of these depending on the scale of the installation

For small scale installations many of these databases can be combined into a single database.

See CommCare HQ docs for more detailed information on different configurations that CommCare supports.

6.1.9 Configuration

The configuration of PostgreSQL is done via the *Configurating postgresql.yml* environment file.

Basic setup

- all databases hosted in a single PostgreSQL service
- relies on defaults for most of the databases

```
DEFAULT_POSTGRESQL_HOST: db1
dbs:
  form_processing:
    partitions:
      p1:
        shards: [0, 127]
      p2:
        shards: [128, 255]
      p3:
        shards: [256, 383]
      p4:
        shards: [384, 511]
      p5:
        shards: [512, 639]
      p6:
        shards: [640, 767]
      p7:
        shards: [768, 895]
      p8:
        shards: [896, 1023]
```

Separate database servers

- `commcarehq`, `formplayer`, `commcarehq_ucr`, `commcarehq_synclogs` hosted on `db1`
- form processing proxy DB (`commcarehq_proxy`) hosted on `plproxy0`
- form processing shard databases split between `pgshard0` and `pgshard1`

```
dbs:
  main:
    host: db1
  formplayer:
    host: db1
  ucr:
    host: db1
  synclogs:
    host: db1
  form_processing:
    proxy:
      host: plproxy0
    partitions:
      p1:
        host: pgshard0
        shards: [0, 127]
      p2:
        host: pgshard0
        shards: [128, 255]
      p3:
        host: pgshard0
        shards: [256, 383]
      p4:
        host: pgshard0
        shards: [384, 511]
      p5:
        host: pgshard1
        shards: [512, 639]
      p6:
        host: pgshard1
        shards: [640, 767]
      p7:
        host: pgshard1
        shards: [768, 895]
      p8:
        host: pgshard1
        shards: [896, 1023]
```

6.2 BlobDB

6.2.1 Migrate from File System backend to an S3 compatible backend

This can be to Riak CS, Minio, S3 or any S3 compatible service.

Ensure that the S3 endpoint is up and accessible

If you are running the service locally make sure it is fully setup. If you are using S3 ensure that you have configured the correct access to allow connections from the IPs where CommCare is running.

Send new writes to the S3 endpoint

1. Update settings in `environments/<env>/public.yml`:
    - `localsettings.BLOB_DB_MIGRATING_FROM_FS_TO_S3: True`
    - `s3_blob_db_enabled: yes`
    - `s3_blob_db_url: "<Endpoint URL e.g. https://s3.amazonaws.com>"`
    - `s3_blob_db_s3_bucket: '<bucket name>'`
2. Add/update settings in `environments/<env>/vault.yml`
    - `secrets.S3_ACCESS_KEY`
    - `secrets.S3_SECRET_KEY`
3. Deploy localsettings .. code-block:: bash

       commcare-cloud <env> update-config
4. Restart CommCare services .. code-block:: bash

       commcare-cloud <env> service commcare resetart

{% include_relative _blobdb_backfill.md %}

Flip to just the new backend

1. Make sure you've run the steps above to move all data to the new backend.
2. Update `environments/<env>/public.yml`
    - Remove `localsettings.BLOB_DB_MIGRATING_FROM_FS_TO_S3`
3. Deploy localsettings with .. code-block:

```
cchq <env> update-config
```

4. Restart CommCare services .. code-block:: bash

       commcare-cloud <env> service commcare resetart

6.2.2 Migrate from one S3 backend to another

This can be from one Riak CS cluster to another, or from a Riak CS cluster to Amazon S3, for example.

Send new writes to the new S3 endpoint

1. If the new endpoint is a riakcs cluster, add `[riakcs_new]` hosts (of new riak cluster) to inventory. (Otherwise skip this step.)
2. Add `localsettings.BLOB_DB_MIGRATING_FROM_S3_TO_S3: True` in `environments/<env>/public.yml`
3. Add/update settings in `environments/<env>/vault.yml`
   - `secrets.S3_ACCESS_KEY` (new cluster)
   - `secrets.S3_SECRET_KEY` (new cluster)
   - `secrets.OLD_S3_BLOB_DB_ACCESS_KEY` (old cluster)
   - `secrets.OLD_S3_BLOB_DB_SECRET_KEY` (old cluster)
4. If the new endpoint is a Riak CS cluster, deploy proxy. This will leave the proxy site for the old endpoint listing to port 8080, and add a new one listening to 8081. (If migrating to Amazon S3, can skip.) .. code-block:: bash

   commcare-cloud <env> ansible-playbook deploy_proxy.yml
5. Deploy localsettings .. code-block:: bash

   commcare-cloud <env> update-config

   During this deploy hosts with old localsettings will continue to talk to old riak cluster on port 8080.

{% include_relative _blobdb_backfill.md %}

Flip to just the new backend

1. Make sure you've run the steps above to move all data to the new backend.
2. Move `[riakcs_new]` hosts to `[riakcs]` (and delete old hosts) in inventory
3. Update `environments/<env>/public.yml`
   - Remove `localsettings.BLOB_DB_MIGRATING_FROM_S3_TO_S3: True`
   - Add `localsettings.TEMP_RIAKCS_PROXY: True`
4. Deploy proxy with .. code-block:

   ```
   cchq <env> ansible-deploy deploy_proxy.yml --tags=nginx_sites
   ```

   You should see both ports (8080 and 8081) now route to the new blobdb backend.
5. Deploy localsettings with .. code-block:

   ```
   cchq <env> update-config
   ```

   During this deploy hosts with old localsettings will continue to talk to new riak cluster on port 8081, and once updated will talk to new riak cluster proxied on port 8080. There is a slight chance of MigratingBlobDB failover from new to new, but this should be rare and benign.
6. Remove `localsettings.TEMP_RIAKCS_PROXY: True` from `environments/<env>/public.yml`
7. Deploy proxy again with .. code-block:

   ```
   cchq <env> ansible-deploy deploy_proxy.yml --tags=nginx_sites
   ```

   You should now see only 8080 route to the new blobdb backend, with the configuration for 8081 being removed.

6.2.3 Back-fill all existing data

After things are set up to read from the new backend and fall back to the old backend, you'll need to run some commands to migrate all existing data from the old backend to the new backend.

First, make sure that the directory you'll use for logs exists:

```
cchq <env> run-shell-command django_manage 'mkdir /opt/data/blobdb-migration-logs; chown␣
↪cchq:cchq /opt/data/blobdb-migration-logs' -b
```

Then run the migration in a tmux (or screen) with these good defaults options (you can tweak the options if you know what you're doing):

```
cchq <env> django-manage --tmux run_blob_migration migrate_backend --log-dir=/opt/data/
↪blobdb-migration-logs --chunk-size=1000 --num-workers=15
```

At the end, you may get a message saying that some blobs are missing and a link to a log file that contains the info about which ones. (If you don't get this message, congratulations! You can skip this step.) Run the following command to check all the blobs in the file to get more info about their status (the last arg is the log file that was printed out above):

```
cchq <env> django-manage --tmux check_blob_logs /opt/data/blobdb-migration-logs/migrate_
↪backend-blob-migration-<timestamp>.txt
```

The output will look something like this:

```
tempfile: checked 2342 records
  Found in new db: 0
  Found in old db: 0
  Not referenced: 0
  Not found: 2342
XFormInstance: checked 42 records
  Found in new db: 0
  Found in old db: 0
  Not referenced: 21
  Not found: 21
```

Legend:

- `Not referenced` is OK. It means that the blob that was said to be "missing" isn't in the blobdb but also isn't referenced by its parent object anymore (this is only meaningful if `BlobMeta.parent_id` is a couch identifier that can be used to lookup the parent object), so it was likely deleted while the migration was running.
- `Not found` means that the missing blob is still referenced in blob metadata, but not found in either the old backend or new backend.
- `Found in (new|old) db` means that actually it is present in one of the backends, unlike originally reported. Re-run the `check_blob_logs` command with `--migrate` to migrate items "Found in old db".

6.2.4 Usage in CommCare

The BlobDB refers to the internal service used by CommCare to store binary data or blobs. The interal API is abstracted from the backend service allowing different backends to be used. Currently supported backends are:

- File system
  - This backend is used for small CommCare deployments where all CommCare services can be run on a single VM / server.
- S3 compatible service (S3, OpenStack Swift, Minio, Riak CS etc.)
  - For larger deployments an Object Storage service is required to allow access to the data from different VMs.

The BlobDB is made up of 2 components:

- PostgreSQL metadata store
  - This keeps track of the object key and it's association to the relevant CommCare models.
- Key based storage service
  - This the actual service where the data is stored. Data can be retrieved by using the object key.

Examples of data that is stored in the BlobDB:

- Form XML
- Form attachments e.g. images
- Application multimedia
- Data exports
- Temporary downloads / uploads

6.2.5 Migrating from one BlobDB backend to another

It is possible to migrate from one backend to another. The process is described in the following documents:

- *Migrate from one S3 backend to another*
- *Migrate from File System backend to an S3 compatible backend*

6.3 Nginx

6.3.1 SSL certificate setup for nginx

CommCare uses LetsEncrypt as the default certificate authority to generate SSL certificates for nginx.

commcare-cloud uses a combination of ansible and certbot to renew our Letsencrypt certificates. Ansible is used to configure the Nginx and certbot. Afterwards Certbot is used for the certificate automation.

Use of Ansible

- Installation and configuration of Certbot.
- Configure Nginx configuration files.
- Creation of directories used for http-01 challenge.
- Creating links to Certbot's fullchain.pem and privkey.pem files

Use of Certbot

- Getting certificate for the first time.
- Renewal of certificate 1 month before the expiry.
- Moving symlinks to latest certificate and private key.

Monitoring

The expiry of certificates is monitored via external monitoring tools such as Datadog or Pingdom.

Procedure to configure SSL for a new environment

:raw-html-m2r:`<small>Note from author**: I haven't tried this many times in a row and noted/fixed all the kinks, so there may be something missing here, but those are the general steps at this point. It would be lovely to make it so that it happened on the first setup, but we're not quite there yet. If there are any errors or gaps, a github issue or pull requests would be much appreciated.</small>`**

1. Set up site without HTTPS

In `proxy.yml`:

- `fake_ssl_cert: yes`
- `letsencrypt_cchq_ssl: yes`
- set `nginx_combined_cert_value` and `nginx_key_value` to `null` (or remove them)

and run full stack deploy

```
commcare-cloud <env> bootstrap-users
commcare-cloud <env> deploy-stack --skip-check -e 'CCHQ_IS_FRESH_INSTALL=1'
commcare-cloud <env> deploy commcare --set ignore_kafka_checkpoint_warning=true
```

Note: if you already have a running site with a previous cert, you can just skip this step.

2. Request a letsencrypt cert

Run the playbook to request a letsencrypt cert:

```
cchq <env> ansible-playbook letsencrypt_cert.yml --skip-check
```

3. Update settings to take advantage of new certs

In `proxy.yml`:

- set `fake_ssl_cert` to `no`

and deploy proxy again.

```
cchq <env> ansible-playbook deploy_proxy.yml
```

6.3.2 Migrating Nginx

1. Install and configure nginx on the new node
   - Add the server in inventory and assign it the `proxy` group and `cas_proxy`(For ICDS)
   - Run `commcare-cloud <env> ap deploy_shared_dir.yml --tags=nfs --limit=shared_dir_host`
   - Run `commcare-cloud <env> ansible-playbook letsencrypt_cert.yml`
   - Run `deploy-stack` for the server.
   - Do a deploy `cchq <env> deploy`
2. Ensure that any files being served directly by nginx are present and identical to the files on the current node
   - Copy static content from live proxy `/home/cchq/www/<env>/current/staticfiles` to new server
   - Copy any other static site content from the live proxy
   - Setup SSL certificates
     - Copy Letsencrypt Configuration dir from live proxy `/etc/letsencrypt` to new server
3. QA
   - Do a test if it's working by editing local `/etc/hosts` file.
4. Switch traffic to the new proxy.
5. Post steps
   - Replace the old server with the new server in the `staticfiles` inventory group.
   - Confirm that the SSL certificate can be renewed correctly by running `certbot renew --dry-run`
   - Run a code deploy to ensure that the CommCare staticfile process is working correctly with the new proxy.

6.3.3 Usage in CommCare

Nginx nginx [engine x] is an HTTP and reverse proxy server. It's primary use in CommCare is as a reverse proxy and load balancer however we also use it for serving static content, content caching and rate limiting.

6.4 Kafka

Table of Contents

- *Kafka*
  - *Resources*
  - *Dependancies*
  - *Setup*
  - *Expanding the cluster*
  - *Useful commands*
  - *Upgrading Kafka*

https://kafka.apache.org/: Kafka is used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant and wicked fast. Kafka is used by CommCare to provide stream processing capabilities for data changes. Changes to certain data models, such as cases and forms, are written to Kafka. Separate processes then read the changes from Kafka and process (pillows) them into Elasticsearch or other secondary databases.

6.4.1 Resources

- Upgrade guide
- Command line tools

6.4.2 Dependancies

- Java
- Zookeeper

6.4.3 Setup

Kafka is installed on each host in the *kafka* inventory group and Zookeeper is installed on each host in the *Zookeeper* inventory group.

6.4.4 Expanding the cluster

1. Add new host to the inventory and install Kafka

```
$ commcare-cloud <env> deploy-stack --limit=<new kafka host>
```

2. Update localsettings

```
$ commcare-cloud <env> update-config
```

3. Move partitions to new node

Follow the steps outlined *below*.

6.4.5 Useful commands

All of the command below assume they are being run from the `/opt/kafka/bin/` path.

Show topic configuration

Note: Use below command when the `kafka version is < 3.x`. The `--zookeeper` argument is removed from 3.x.

```
$ ./kafka-topics.sh --describe --zookeeper=<zookeeper host>:2181 --topic
↪<topic>
```

Note: Use below command when the `kafka version is >= 3.x`.

```
$ ./kafka-topics.sh --describe --bootstrap-server=<kafka host>:9092 --topic
↪<topic>
```

Add new partitions to topic

N is the total number of partitions the topic should have

Note: Use below command when the `kafka version is < 3.x`. The `--zookeeper` argument is removed from 3.x.

```
$ ./kafka-topics.sh --alter --zookeeper=<zookeeper host>:2181 --topic <topci> -
↪-partitions N
```

Note: Use below command when the `kafka version is >= 3.x`.

```
$ ./kafka-topics.sh --alter --bootstrap-server=<kafka host>:9092 --topic
↪<topci> --partitions N
```

Note: Adding partitions to a topic should be done in conjunction with updating the CommCare Pillowtop process configurations as described in the CommCare docs.

Move partitions

NOTE: This can be done while all services are online

1. Create the list of topics to rebalance

```
$ cat topics.json
{
  "topics": [{"topic": "case-sql"},{"topic": "form-sql"}],
  "version": 1
}
```

2. Generate the reassignments

Note: Use below command when the `kafka version is < 3.x`. The `--zookeeper` argument is removed from 3.x.

```
$ /opt/kafka/bin/kafka-reassign-partitions.sh --zookeeper=localhost:2181 --
↪broker-list "0,1,2" --topics-to-move-json-file topics.json --generate
```

Note: Use below command when the `kafka version is >= 3.x`.

```
$ /opt/kafka/bin/kafka-reassign-partitions.sh --bootstrap-
↪server=localhost:9092 --broker-list "0,1,2" --topics-to-move-json-file␣
↪topics.json --generate
```

Output:

1. Copy the proposed reassignment configuration to a JSON file and verify / update as required

   `replicas` refers to the broker IDs that the partition should appear on. In the example below this will put the `("case", 0)` partition on broker 0 (with no replicas).

```
$ cat partitions-to-move.json
{
  "version":1,
  "partitions":[
    {"topic":"case","partition":0,"replicas":[0]},
    ...
  ]
}
```

2. Reassign the partitions and verify the change:

Note: Use below command when the `kafka version is < 3.x`. The `--zookeeper` argument is removed from 3.x.

```
$ ./kafka-reassign-partitions.sh --zookeeper=localhost:2181 --reassignment-
↪json-file partitions-to-move.json --execute

$ ./kafka-reassign-partitions.sh --zookeeper=localhost:2181 --reassignment-
↪json-file partitions-to-move.json --verify
```

Note: Use below command when the `kafka version is >= 3.x`.

```
$ ./kafka-reassign-partitions.sh --bootstrap-server=localhost:9092 --
↪reassignment-json-file partitions-to-move.json --execute
```

(continues on next page)

(continued from previous page)

```
$ ./kafka-reassign-partitions.sh --bootstrap-server=localhost:9092 --
↪reassignment-json-file partitions-to-move.json --verify
```

See https://kafka.apache.org/documentation.html#basic_ops_cluster_expansion for more details.

Replication

For setting up the replication on existing topic we make use of a helper script which has the following capabilities:

- increase replication for existing topics
- decrease replication factor for existing topics
- remove all replicas from a particular broker so it can be decomissioned
- balance leaders

For details on how to use this tool please see kafka-reassign-tool

6.4.6 Upgrading Kafka

```
KAFKA-VERSION: Version of Kafka being upgraded to
KAFKA-SCALA-VERSION: Version required by KAFKA-VERSION ( Can be found `here
↪<https://kafka.apache.org/downloads>`_) .
KAFKA_INTER_BROKER_PROTOCOL_VERSION: Maps to Kafka's inter.broker.protocol.
↪version. If you have a cluster that runs brokers with different Kafka␣
↪versions make sure they communicate with the same protocol version.
KAFKA_LOG_MESSAGE_FORMAT_VERSION: Maps to Kafka's log.message.format.version.␣
↪Specifies the protocol version with which your cluster communicates with its␣
↪consumers.
```

Refer to Kafka Upgrade documentation for more details.

1. Ensure that the Kafka config is up to date

   ```
   $ cchq <env> ap deploy_kafka.yml
   ```

2. Update the Kafka version number and Scala version in `public.yml`. For right Scala version please refer the *Kafka documentation <https://kafka.apache.org/downloads>*.

   environments/`<env>`/public.yml

   ```
   kafka_version: <KAFKA-VERSION>
   kafka_scala_version: <KAFKA-SCALA-VERSION>
   ```

3. Upgrade the Kafka binaries and config

   ```
   $ cchq <env> ap deploy_kafka.yml
   ```

4. Upgrade the brokers one at a time Once you have done so, the brokers will be running the latest version and you can verify that the cluster's behavior and performance meets expectations. It is still possible to downgrade at this point if there are any problems.

5. Update Kafka config:

environments/`<env>`/public.yml

```
kafka_inter_broker_protocol_version: <KAFKA_INTER_BROKER_PROTOCOL_VERSION>
```

```
$ cchq <env> ap deploy_kafka.yml
```

6. Update Kafka config (again):

   environments/`<env>`/public.yml

```
kafka_log_message_format_version: <KAFKA_LOG_MESSAGE_FORMAT_VERSION>
```

```
$ cchq <env> ap deploy_kafka.yml
```

6.5 Pillowtop

Table of Contents

- *Pillowtop*
  - *Usage in CommCare*
  - *Splitting a pillow*

Pillowtop is an internal framework built in CommCare which is used for asynchronous stream processing of data.

A *pillow* is a class build in the *pillowtop* framework. A pillow is a subscriber to a change feed. When a change is published the pillow receives the document, performs some calculation or transform, and publishes it to another database.

In general a *change feed* refers to a Kakfa topic or topics but could also be a CouchDB change feed.

More information on the architecture and code structure are available in the CommCare documentation:

- Change Feeds
- Pillows

6.5.1 Usage in CommCare

CommCare uses pillows to populate its secondary databases. These databases are used for reporting and also back some of the CommCare features like APIs.

These databases are:

- Elasticsearch
- SQL custom reporting tables

6.5.2 Splitting a pillow

In some cases a pillow may contain multiple processors. It is sometimes desirable to split up the processors into individual OS processes. The most compelling reason to do this is if one of the processors is much slower than the others. In this case having the slow processor separated allows the other's to process at a faster pace. It may also be possible to deploy additional processing capacity for the slow one.

The the following steps we will be splitting the `FormSubmissionMetadataTrackerProcessor` out from the `xform-pillow`.

The `xform-pillow` has multiple processors as can be seen from the CommCare docs. The `FormSubmissionMetadataTrackerProcessor` can be disabled by setting `RUN_FORM_META_PILLOW` to `False` in the Django settings file. We can also see that the `FormSubmissionMetadataTrackerProcessor` is used by the `FormSubmissionMetadataTrackerPillow`.

So in order to split the `FormSubmissionMetadataTrackerProcessor` into its own pillow process we need to do the following:

1. Update the environment configuration
   - Add the new pillow to `<env>/app-processes.yml`

     ```
     pillows:
       'pillow_host_name':
         FormSubmissionMetadataTrackerPillow:
           num_processes: 3
     ```

   - Update the `RUN_FORM_META_PILLOW` to disable the processor in the current pillow:

     ```
     localsettings:
       RUN_FORM_META_PILLOW: False
     ```

2. Stop the current pillow

   ```
   cchq <env> service pillowtop stop --only xform-pillow
   ```

3. Create new checkpoints for the new pillow

   ```
   cchq <env> django-manage --tmux split_pillow_checkpoints xform-pillow␣
   ↪FormSubmissionMetadataTrackerPillow
   ```

   Note: `--tmux` is necessary to allocate a terminal for the command which allows you to respond to any confirmation prompts.

4. Deploy the new pillow

   ```
   cchq <env> update-supervisor-confs
   ```

5. Ensure all the pillows are started

   ```
   cchq <env> service pillowtop start
   ```

6.6 RabbitMQ

6.6.1 Usage in CommCare

RabbitMQ is a message broker which supports publish/subscribe workflows. RabbitMQ groups exchanges, queues, and permissions into virtual hosts. For our setup in production, the virtual host is always "commcarehq". Locally you might be using the default virtual host of "/".

6.6.2 Guides

- services/rabbitmq/upgrade:Upgrading RabbitMQ

6.7 Redis

6.7.1 Usage in CommCare

Redis is an open source, in-memory data structure store. CommCare uses Redis for caching and locking.

6.7.2 Guides

- services/redis/redis_cluster:Redis Cluster

6.7.3 Tools

- Redis Traffic Stats

6.7.4 Configuration recommendations

Disk

You should allocate at least 3x as much disk for redis data as the redis `maxmemory` setting.

If you define a `redis_maxmemory` variable in your environment's public.yml then that will be the value. Otherwise it will be half of the total memory of the machine.

So for example if `redis_maxmemory` is not set and you're running redis on an 8GB machine, then redis's `maxmemory` setting will be 4GB and you should allocate at least 4GB x 3 = 12GB of disk for redis data (in addition to whatever other disk space needed for the OS, logs, etc.). This will allow enough room for redis's AOF (data persistence file), whose rewrite process makes it oscillate between at most `maxmemory` and at most `3 x maxmemory` in a saw-tooth fashion.

6.8 Set up Bitly for generating app codes

To enable generating app shortcodes to install builds on CommCare mobile, you will need a Bitly API key.

You should add these in the ansible vault file as follows:

vault.yml

```
localsettings_private:
  BITLY_OAUTH_TOKEN: ''
```

6.9 Keepalived

Keepalived is used for IP failover between two servers. It adds facilities for load balancing and high-availability to Linux-based infrastructures. Keepalived works on VRRP (Virtual Router Redundancy Protocol) protocol to make IPs highly available .

The VRRP protocol ensures that one of participating nodes is master. The backup node(s) listens for multicast packets from a node with a higher priority. If the backup node fails to receive VRRP advertisements for a period longer than three times of the advertisement timer, the backup node takes the master state and assigns the configured IP(s) to itself. In case there are more than one backup nodes with the same priority, the one with the highest IP wins the election.

Note: No fencing mechanism is available in Keepalived. If two participating nodes don't see each other,both will have the master state and both will carry the same IP(s). In our Current infrastructure we are using it for couchdb proxy failover in ICDS environment. We have plans to implement it more places where we have proxies setup.

6.9.1 Operational Notes:-

1. check keepalived status

   ```
   $ systemctl status keepalived
   ```

2. **know which one is the master**

   Check the status/logs and you'll see the log lines like this

   `Sep 12 03:25:36 MUMGCCWCDPRDCDV09 Keepalived_vrrp[30570]: VRRP_Instance(VI_1) Entering BACKUP STATE` Check the virtual IP listed in `/etc/keepalived/keepalived.conf`, verify if this IP address is assigned to the interface of the server.

3. where are the logs

   Keepalived logs to journald

   ```
   $ journalctl -u keepalived
   ```

4. if we restart haproxy will that trigger a failover? (only on master node)

   ```
   vrrp_script chk_service {
       script "pgrep haproxy"
       interval 2
   }
   ```

From the config above it check for running process in every two second interval.if restart took longer than this, it will trigger failover

1. what's the process for taking haproxy offline e.g. during maintenance
    - If we are performing Maintenance on Backup Node
        - No Action
    - If we are performing Maintenance on Master Node
        - Stop haproxy and verify if Backup node is transitioned to master state from logs of Backup node (This is optional for just to be safe otherwise it should transition automatic once the haproxy stops on master node)
    - If we are Performing on both nodes at the same time.
        - Not much anyone can do

Note:- All the nodes will go back to their desired state once the maintenance is over.

CHAPTER

SEVEN

BACKUPS AND DISASTER RECOVERY

This section describes how to configure backups and restore method for each database service used by CommCare HQ instance.

7.1 Backup and Restore

Table of Contents

- *Backup and Restore*
  - *Warning*
  - *Backup to Amazon S3 or a compatible service*
  - *PostgreSQL Backups*
  - *CouchDB backups*
  - *BlobDB Backups*
  - *Elasticsearch Snapshots*

This page describes some of the backup options that can be accessed through CommCare cloud.

7.1.1 Warning

You should read this section carefully and understand what each of these settings does. Backups are system dependent, and you should convince yourself that they are working correctly and that you are properly able to restore from them before something bad happens.

Each primary data-store that CommCare HQ uses can have backups turned on or off based settings in *public.yml* or the vault file. All settings mentioned below are to be placed in `public.yml` unless otherwise specified.

After making changes to these settings you will need to run:

```
$ commcare-cloud <env> deploy-stack --tags=backups
```

7.1.2 Backup to Amazon S3 or a compatible service

`commcare-cloud` has the ability to upload all backups automatically for storage on Amazon S3 or an S3-compatible alternative. Each service's backup has a specific setting that needs to be enabled for this to happen, as detailed below.

S3 credentials

In order to use this service, you will need to add your S3 credentials to the `localsettings_private` section of your **vault file**:

- `AMAZON_S3_ACCESS_KEY`: Your aws access key id
- `AMAZON_S3_SECRET_KEY`: Your aws secret access key

Even though these settings have the word `AMAZON` in them, you should use use the credentials of your S3-compatible hosting provider.

Endpoints

We use boto3 to upload data to Amazon S3 or a compatible service.

- `aws_endpoint`: The endpoint to use. Add this setting if you are using an S3-compatible service that isn't AWS.
- `aws_region`: The Amazon AWS region to send data to. (Amazon S3 only - this changes the default aws-endpoint to the region-specific endpoint).
- `aws_versioning_enabled`: (*true* or *false*) Set this to *true* if the AWS endpoint you are using automatically stores old versions of the same file (Amazon S3 does this). If this is set to *false*, files will be uploaded to your S3-compatible bucket with a date and timestamp in the filename, creating a new file each time. (Default: `true`)

Receiving email alerts if check fails

There is a script that can run to check for the presence of recent backups uploaded to S3, and it currently supports blobdb, couch, and postgres. To enable it, configure the following variable in public.yml:

```
check_s3_backups_email: backup-alerts@example.com
```

It's your responsibility to test that you receive these emails when recent backups are missing in S3 and that the emails don't go to you spam folder before treating the absence of alerts as a positive signal. In addition to sending an email when there's an error, it will place a file called `s3_backup_status.txt` inside the backup dir for each service. You can check for the presence of that file and its last modified date when looking for evidence that the backup check is running correctly.

7.1.3 PostgreSQL Backups

PostgreSQL backups are made daily and weekly by default and can be made hourly optionally. Old backups are deleted from the local system.

- *postgresql_backup_dir*: The directory to write the PostgreSQL backups to. (Default: `/opt/data/backups/postgresql`)
- The `backup_postgres` setting has a few options. You should understand the tradeoffs of each of these settings and know how to restore from the resulting backup files.
  - `plain` - uses the pg_basebackup command to write a backup to the `postgresql_backup_dir`.

  - `dump` - uses the pg_dumpall command to write a dump of the database to the `postgresql_backup_dir`.
- *postgresql_backup_days*: The number of days to keep daily backups (Default: 1)
- *postgresql_backup_weeks*: The number of weeks to keep weekly backups (Default: 1)
- *postgres_backup_hourly*: Boolean value to enable or disable hourly backups. (Default: false)
- *postgresql_backup_hours*: The number of hours to keep hourly backups (Default: 1).

Enabling S3 backups for PostgreSQL

After *adding your credentials* to the vault file, set:

- `postgres_s3: True`
- *postgres_snapshot_bucket*: The name of the S3 bucket to save postgres backups to (Default: `dimagi-<env>-postgres-backups`).

Restoring PostgreSQL Backups

You should first stop all CommCare HQ services:

```
$ commcare-cloud <env> downtime start
$ commcare-cloud <env> service postgresql stop
```

Restoring from backup depends on the type of backup made.

plain (pg_basebackup) without S3

If you are using a `pg_basebackup`, you should follow these instructions. The latest daily backup should be in the directory specified in `postgresql_backup_dir`, above.

For example, you can follow a process similar to this one:

- You will need to run commands as the `postgres` user:

```
$ su - ansible
# enter ansible user password from vault file
$ sudo -u postgres bash
# enter ansible user password again. You will now be acting as the postgres user
```

- Find the list of current backups and choose the one you want to restore from, for e.g.:

```
$ ls -la /opt/data/backups/postgresql # or whatever your postgres backup directory␣
↪is set to
total 3246728
drwxr-xr-x 2 postgres postgres      4096 Jul  8 00:03 .
drwxr-xr-x 5 root     root          4096 Feb  6  2018 ..
-rw-rw-r-- 1 postgres postgres 678073716 Jul  6 00:03 postgres_<env>_daily_2019_07_
↪06.gz
-rw-rw-r-- 1 postgres postgres 624431164 Jun 23 00:03 postgres_<env>_weekly_2019_06_
↪23.gz
```

- Uncompress the one you want:

```
$ tar -xjf /opt/data/backups/postgresql/postgres_<env>_daily_2019_07_06.gz -C /opt/
↪data/backups/postgresql
```

- [Optional] Make a copy of the current data directory, for eg:

```
$ tar -czvf /opt/data/backups/postgresql/postgres_data_before_restore.tar.gz /opt/
↪data/postgresql/9.6/main
```

- Copy backup data to the postgres data directory. This will overwrite all the data in this directory.

```
$ rsync -avz --delete /opt/data/backups/postgresql/postgres_<env>_daily_2019_07_06 /
↪opt/data/postgresql/9.6/main
```

- Restart Postgres and services, from the control machine, e.g.:

```
$ commcare-cloud <env> service postgresql start
```

plain (pg_basebackup) with S3

If you have S3 backups enabled there is a restore script that was installed when the system was installed.

On the PostgreSQL machine:

- Become the root user

```
$ su - ansible
# enter ansible user password from vault file
$ sudo -u root bash
# enter ansible user password again. You will now be acting as the root user
```

- Run the restore script after finding the backup you want to restore from S3

```
$ restore_from_backup <name of backup file>
```

Note: this script will not make a copy of the current data directory and should be used with caution. You should know and understand what this script does before running it.

dump (pg_dumpall)

You can follow these instructions to restore from a dump. You will need to have a new database set up with a root user as described in the instructions.

- Ensure the file you are restoring from is readable by the postgres user. By default, `commcare-cloud` will make backups into `/opt/data/backups/postgresql/` as `.gz` zipped archives. Choose one of these files as the source of your backup.
- Become the postgres user

```
$ su - ansible
# enter ansible user password from vault file
$ sudo -u postgres bash
# enter ansible user password again. You will now be acting as the postgres user
```

- Extract the backup and pipe it to the `psql` command to restore the data contained in the backup. The name of the default postgres database is `commcarehq`:

```
$ gunzip -c <path to backup file> | psql commcarehq
```

7.1.4 CouchDB backups

CouchDB backups are made daily and weekly and optionally hourly. Old backups are deleted from the system.

- `backup_couch:` `True` to enable couchdb backups (Default: `False`)
- `couch_s3:` `True` to enable sending couchdb backups to your S3 provider (Default: `False`)
- *couch_backup_dir*: the directory to save backups in (Default: `/opt/data/backups/couchdb2`)
- *couchdb_backup_days*: The number of days to keep daily backups (Default: 1)
- *couchdb_backup_weeks*: The number of weeks to keep weekly backups (Default: 1)
- *couch_backup_hourly*: Boolean value to enable or disable hourly backups. (Default: false)
- *couchdb_backup_hours*: The number of hours to keep hourly backups (Default: 1).

CouchDB backups create a compressed version of the couchdb data directory.

Restoring CouchDB backups (on a single node cluster)

Make sure that you are starting with a fresh install of couchdb.

- First, become the couchdb user: .. code-block:: bash

  $ su - ansible # enter ansible user password from vault file $ sudo -u couchdb bash # enter ansible user password again. You will now be acting as the couchdb user

- [Optional] Copy the contents of the current couchdb directory in case anything goes wrong. From the couchdb machine:

```
$ tar -czvf /opt/data/backups/couchdb2/couchdb_data_before_restore.tar.gz -C /opt/
↪data/couchdb2/ .
```

- Locate the compressed backup file that you want to restore. If this is stored somewhere remotely, you should put it on this machine in a place accessible to the `couchdb` user. By default, couchdb backups live in `/opt/data/backups/couchdb2`.
- Run the restore script:

```
   $ restore_couchdb_backup.sh <path to backup>

This script will extract the backup file to the default couchdb backup location,␣
↪copy this data to the couchdb data directory, the updates the couchdb shards with␣
↪the current machine's IP addresses.

During this process you will be asked for the ansible user's password in order to␣
↪stop and start the couchdb service.

**Note**\ : This backup script will only work for a single-node cluster.
```

- As your regular user, ensure the couchdb service is now running:

```
$ commcare-cloud <env> django-manage check_services
```

7.1.5 BlobDB Backups

The `blobdb` is our binary data store.

- `backup_blobdb: True`: to enable blobdb backups
- `blobdb_s3: True`: to enable sending blobdb backups to S3
- *blobdb_backup_dir*: the directory to write blobdb backups to (Default: `/opt/data/backups/blobdb`)
- *blobdb_backup_days*: the number of days to keep daily backups (Default: 2)
- *blobdb_backup_weeks*: the number of weeks to keep weekly backups (Default: 2)
- *blobdb_backup_hourly*: Boolean value to enable or disable hourly backups. (Default: false)
- *blobdb_backup_hours*: The number of hours to keep hourly backups (Default: 1).

BlobDB backups create a compressed version of the blobdb data directory.

Restoring BlobDB Backups

The BlobDB restore process depends on what BlobDB system you're using.

- If you're using the default file system BlobDB, the restore process is the same as the couchdb restore process in that it involves extracting the backed up data to the data directory.
- If you're using some other (distributed) system you should follow that service's provided instructions on restoration.

The file system BlobDB restore process will be explained below.

- Become the *cchq* user

```
$ sudo -iu cchq
```

- Now we need to extract the backup data. The BlobDB backups live in the */opt/data/backups/blobdb* directory by default (if you have specified a different path in the *public.yml* file, it will be there instead).

```
$ tar -xf /opt/data/backups/blobdb/blobdb_<version>.gz -C /opt/data/backups/blobdb
```

- Move the data to the */opt/data/blobdb/* directory.

```
$ rsync -avz --delete /opt/data/backups/blobdb/blobdb_<version> /opt/data/blobdb/
```

7.1.6 Elasticsearch Snapshots

While it is possible to backup Elasticsearch data, it isn't always necessary as this is not a primary data store and can be rebuilt from primary sources. If Elasticsearch data is lost or deleted in entirety, it will be recreated when *Deploying CommCare HQ code changes*.

However, you may still back-up Elasticsearch using Elasticsearch Snapshots directly to S3 or locally. The rest of this section assumes an understanding of that documentation page.

- `backup_es_s3: True`: to create snapshots and send them directly to S3 (not stored locally)

- `es_local_repo: True`: to save snapshots locally (not sent to S3)
- `es_repository_name`: the name to give to the snapshot respository

Both of those settings are **mutually exclusive**. There is currently no way to create snapshots to be saved locally and sent to S3 at the same time.

Restoring Elasticsearch Snapshots

You can restore snapshots by following the instructions given by Elasticsearch

7.2 ElasticSearch Backup on Swift API

We normally do the backup of ElasticSearch using ElasticSearch backup plugin which allows us to take backup on external seervices compatible with S3. In Few cases where S3 is not availble we can sort to other solutions. This documentaion details the same process for backing up on Swift API of OpenStack Plugin used : https://github.com/BigDataBoutique/elasticsearch-repository-swift

7.2.1 Configuring and Testing.

To install the plugin on the ansible server.

- install the plugin using elasticsearch plugin binary. ```bash

/opt/elasticsearch-1.7.6/bin/plugin install org.wikimedia.elasticsearch.swift/swift-repository-plugin/1.7.0```

```
* To create a Repo for the sanpshot
```bash

curl -XPUT 'http://<ip-address>:9200/_snapshot/<env>_es_snapshot' -d '{
>         "type": "swift",
>         "settings": {
>             "swift_url": "https://<aurl-address>/auth/v1.0/",
>             "swift_container": "nameofthecontainer",
>             "swift_username": "XXXXXX",
>             "swift_password": "XXXXX",
>             "swift_authmethod": ""
>         }
>     }'
{"acknowledged":true}
```

- To take a snapshot ```bash

curl -X PUT "localhost:9200/_snapshot/**:raw-html-m2r:`<env>`**_es_snapshot/snapshot_1?wait_for_completion=true"

```
* To Verify the snapshot.
```bash

curl -X GET "<ip-address>:9200/_snapshot/<env>_es_snapshot/_all"
```

- To restore a snapshot of date say 2018/10/05. ```bash

curl -X POST "**:raw-html-m2r:`<ip-address>`**:9200/_snapshot/**:raw-html-m2r:`<env>`**_es_snapshot/**:raw-html-m2r:`<env>`**_es_snapshot_2018_10_5/_restore"

```
## Configuring in Ansible
Once you can check that above process is working fine you can proceed with configuring␣
↪the same in Ansible.

Add the following entries in `public.yml` of the environemtn you want to configure.
```bash
# ElasticSearch Backup on Swift API
backup_es_swift: True
swift_container: "nameofthecontainer"
swift_url: https://<aurl-address>/auth/v1.0/
```

Add the follwing line in vault.yml

```
secrets
  swift_username: "XXXXXXXXXXX"
  swift_password: "YYYYYYYYYYY"
```

Deploy elasticsearch

```
cchq <env> anisble-playbook deploy_db.yml --limit=elasticsearch
```

What Does Ansible do.

- Install the Plugin
- Restart ElasticSearch
- Create a snapshot repo
- Copy script to take snapshot
- Create a Cronjob

7.3 Disaster Recovery

Table of Contents

- *Disaster Recovery*
  - *Overview*
  - *Setting up a secondary environment*
  - *Remote Backups*
  - *Database Replication*
  - *Example models*
```

7.3.1 Overview

Disaster Recovery refers to the process of restoring the IT services following an event of outage of IT infrastructure due to a natural or human induced disaster to allow business continuity. This public article from IBM is useful to understand what Disaster Recovery is in general.

A Disaster Recovery solution at a minimum involves

- Establishing Recovery Time and Recovery Point Objectives that meet project requirements.
- Setting up and monitoring remote data backups or replication.
- Creating an active or passive secondary site and the necessary automation to failover and restore to this site rapidly.

Since CommCare Mobile works offline, a disaster at primary infrastructure may not cause an immediate disruption to mobile worker operations. But it definitely impacts all web operations (for e.g. actions based on reports, SMS reminders and integrations) and will soon clog mobile worker operations as well.

In this regard, to ensure continuity of a CommCare deployment following an event of outage, you must have a comprehensive Disaster Recovery Solution. The purpose of this document is not to present any single Disaster Recovery solution, but rather to present the possibilities and components that commcare-cloud provides to enable a Disaster Recovery. The specific solution may vary depending on business needs, IT capacity and DR budget and we recommend you to design a DR solution that meets all of your requirements upfront as part of initial CommCare deployment.

7.3.2 Setting up a secondary environment

To set up a secondary environment, copy the existing environment *config directory* to another directory and use it to *install CommCare HQ*.

If you plan to have a secondary environment identical to primary environment the only file you will need to update is the inventory file with IP addresses from secondary environment.

If not, you can create another environment directory with updated configuration based on the primary config directory.

7.3.3 Remote Backups

The commcare-cloud backup tool provides everything to enable remote backups. It takes daily and weekly backups by default and can be configured to take hourly backups. The tool can send backups to a remote S3 compatible service if configured. The documenation on *Backup and Restore* has the relevant details on how to set up remote backups and restore from backups.

While the remote backups are the minimum requirement to safeguard the data in the event of a disaster it is not enough for a rapid recovery of the CommCare services. Even though, commcare-cloud is able to provide hourly backups it will not be enough to achieve one hour RPO since it is not possible to set up a CommCare environment rapidly in an hour at a secondary site. To enable rapid recovery you must have a secondary environment on standby, set up database replication and a failproof recovery plan and scripts.

7.3.4 Database Replication

Continous Database replication is necessary to enable minimal RPO. While commcare-cloud does not have tooling available to set up continous replication all the databases used in CommCareHQ support replication through various means. To set this up, you should consult database specific documentation and set this up yourself.

Primary Databases

- Postgresql https://www.postgresql.org/docs/current/runtime-config-replication.html
- BlobDB If you are using MinIO https://min.io/product/active-data-replication-for-object-storage
- CouchDB https://docs.couchdb.org/en/stable/replication/intro.html

Secondary Databases

- Elasticsearch https://www.elastic.co/guide/en/cloud-enterprise/2.4/ece-snapshots.html#ece-restore-across-clusters

7.3.5 Example models

Below are some example models for DR.

- Remote backups on secondary environment: You can set up a CommCare HQ environment on secondary infrastructure, keep it up to date with remote backups and keep restore scripts ready.
- Secondary environment with contious data replication: You can setup a fully functioning secondary environment with databases being replicated continously.
- Remote backups on secondary infrastructure (Not a DR): In this all you need is a secondary infrastructure to fall back to and backups being continously sent to this site.

CHAPTER

EIGHT

SECURING COMMCARE HQ

Security is one of the most important things when dealing with web applications. This guide gives high level overview of security considerations to protect the data that is collected using a CommCare HQ instance. Note that this just gives hints in the direction of the security and is in no way exhaustive.

8.1 Introduction

A web application exposed to public internet has many levels where an attack is possible such as application, host operating system, network and even physical levels. In a cloud environment, the cloud service provider handles the security up to operating system. But when hosting on premises all of this responsibility falls upon the team hosting the application.

It is very important to design an end-to-end hosting architecture and processes to maximize security at all levels. In addition, there might be government regulations and legal requirements to be met when hosting locally that the local hosting team may need to be aware of.

Below are few security focused architecture from the industry as a reference to understand the number of considerations when designing a secure hosting environment.

1. https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise/design-zone-security/safe-secure-dc-architecture-guide.pdf
2. https://www.oracle.com/a/ocom/docs/oracle-cloud-infrastructure-security-architecture.pdf
3. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-123.pdf

Below we provide few of such considerations that we recommend at a minimum.

Note: CommCare and its server platform CommCare HQ are Open Source software, primarily developed by Dimagi. These are made available to the community under the terms of its Open Source licensing without warranty.

We regularly undertake security efforts like penetration testing, audits of the software code, and reviews to ensure that the system functionality is sufficient to meet compliance commitments that we make to our commercial customers in providing our SaaS service. We believe that these demonstrate that CommCare can meet very high standards of scrutiny when deployed appropriately.

To best support our community of practice, below we provide security best practices which are common to the security needs of our partners. These materials are provided for reference without warranty for their accuracy or sufficiency to meet a particular standard of security, and do not constitute any commitment from the authors or owners of the code.

8.2 Application Security

Application refers to all the services that are accessible for users mostly through HTTP. In the context of CommCare HQ, it's the CommCare HQ website that is being hosted and any other services that have HTTP endpoints such as Elasticsearch and Celery Flower. Here are few things that must be taken care of to ensure safety of the application services.

1. **Access Management**
   - CommCare HQ has finegrained roles and permissions based access management. When registering users make sure they have appropriate roles and permissions.
   - For users with administrative privileges make sure they set up and use Two Factor Authentication.
2. Refer to `Privacy and Security` section in Project Space Settings and configure as necessary.
3. Be up to date with changes and security updates for CommCare HQ and commcare-cloud by following *Expectations for Ongoing Maintenance*.
4. Make sure to configure SSL certificate using docs at *SSL certificate setup for nginx*.

8.3 Host and Disk Security

1. **Access management** Make sure that access to virtual machines is done using SSH keys and not passwords. Refer to *User Access Management* to know how this is done using commcare-cloud. Implement any other best practices such as enabling access through VPN and logging SSH access etc as necessary.
2. **Data Encrypttion** When CommCare HQ is deployed with commcare-cloud all the drives that store user data are automatically encrypted. If the data is stored anywhere else, it must be made sure that the data is stored only in encrypted drives.
3. **Secrets** All the passwords are stored in the ansible encrypted vault file. Never expose these passwords and store and share the password for the vault file securely.
4. It's recommended to take support contracts for Ubuntu and any other virtualization software.
5. Make sure that there is a process in place to get alerts on security patches for the operating system and other important libraries.

8.4 Network and Physical Security

1. Use VPN to access virtual machines when outside the LAN.
2. Make sure to implement necessary firewall rules to enable restricted access to the virtual machines.
3. If the hosting hardware is shared with other applications alongside CommCare HQ, additional network functionalities may need to implemented to ensure security isolation of the applications.
4. Implement necessary protocols to secure access to the physical servers at the data center or server room.

CHAPTER

NINE

REFERENCE ANNEXURE

This section contains reference sections for various topics related to hosting a CommCare HQ instance.

9.1 CommCare Cloud Reference

commcare-cloud is a python based tool used to automate deployment of CommCare HQ on a given set of servers. For more information on what commcare-cloud is please see *CommCare Cloud Deployment Tool*.

9.1.1 Installation

commcare-cloud can be installed on a local machine or on a remote control machine that's part of the CommCare HQ environment. We recommend installing on a control machine.

Installation using a script

Step 1.

Make sure that you have a non-root user account on the control machine.

Let's say the user is named `admin` and the machine is named `control.example.com`. Start by logging in as your user via SSH:

```
(laptop)$ ssh admin@control.example.com
```

(Did that work? Only type the text starting after the `$` in these examples.)

You should see a prompt like

```
admin@control.example.com:~$
```

Run this command to verify that you are in the home directory of the `admin` user.

```
admin@control.example.com:~$ pwd
/home/admin
```

Step 2.

Pull commcare-cloud source code.

```
admin@control.example.com:~$ git clone https://github.com/dimagi/commcare-cloud.git
```

Verify that created a directory called `commcare-cloud`:

```
admin@control.example.com:~$ ls commcare-cloud/
commcare-cloud-bootstrap  environments  MANIFEST.in   setup.py
control                   fabfile.py    provisioning  src
dev_requirements.txt      git-hooks     README.md     tests
docs                      Makefile      scripts       Vagrantfile
```

If you see something like

```
ls: cannot access commcare-cloud: No such file or directory
```

then the `git clone` command did not run correctly. Make sure you have git installed and run it again with `--verbose` to give more logging output.

If you see

```
fatal: destination path 'commcare-cloud' already exists and is not an empty directory.
```

Run the following commands to update the existing commcare-cloud repository

```
admin@control.example.com:~$ cd commcare-cloud
admin@control.example.com:~$ git checkout master
admin@control.example.com:~$ git pull
admin@control.example.com:~$ cd ..
```

Step 3.

Run the install script.

```
admin@control.example.com:~$ source commcare-cloud/control/init.sh
```

and when you see it ask you this:

```
Do you want to have the CommCare Cloud environment setup on login?
(y/n):
```

answer with `y`. This will make `commcare-cloud` available to run every time you log in.

To check that commcare-cloud is now installed, run

```
admin@control.example.com:~$ commcare-cloud -h
usage: commcare-cloud [-h] [--control]

                      {64-test,development,echis,icds,icds-new,pna,production,softlayer,
↪staging,swiss}
                      {bootstrap-users,ansible-playbook,django-manage,aps,tmux,ap,
↪validate-environment-settings,deploy-stack,service,update-supervisor-confs,update-
```

(continues on next page)

(continued from previous page)

```
↪users,ping,migrate_couchdb,lookup,run-module,update-config,mosh,after-reboot,ssh,
↪downtime,fab,update-local-known-hosts,migrate-couchdb,run-shell-command}
                        ...
```

...and then much more help output describing each possible command.

If you get to this point, congratulations! `commcare-cloud` is installed.

Manual Installation

You will need python 3.10 installed to follow these instructions. See installation/2-manual-install:Upgrade to Python 3.10 for instructions on getting it installed on Ubuntu 22.04. Steps for other operating systems may differ.

Setup

Download and run the `control/init.sh` script. This should be run from your home directory:

```
source <(curl -s https://raw.githubusercontent.com/dimagi/commcare-cloud/master/control/
↪init.sh)
```

You will see the following prompt

```
Do you want to have the CommCare Cloud environment setup on login?
(y/n):
```

If you answer 'y' then a line will be added to your .profile that will automatically run `source ~/init-ansible` when you log in, sets up the commcare-cloud environment. Otherwise, you can choose to run `source ~/init-ansible` manually to setup the environment during future sessions.

You may now use `commcare-cloud` or its shorter alias `cchq` whenever you're in the virtualenv.

Manual setup

If you'd rather use your own virtualenv name or a different commcare-cloud repo location, or if the script above did not work.

Setup and activate the virtualenv

NOTE: *The virtualenv name and location may be customized, below example uses ``cchq`` and ``~/.virtualenvs/cchq``. Adjust according to your preferred configuration.*

```
# using venv
python3.10 -m venv ~/.virtualenvs/cchq
source ~/.virtualenvs/cchq/bin/activate

# -- or --

# using pyenv
pyenv virtualenv 3.10 cchq
pyenv activate cchq
```

Install commcare-cloud with pip

```
# IMPORTANT: ensure the virtual environment is activated
git clone https://github.com/dimagi/commcare-cloud.git
cd ./commcare-cloud
pip install --upgrade pip-tools
pip-sync requirements.txt
pip install -e .
manage-commcare-cloud install

# (Optional) To use commcare-cloud (cchq) without needing an active virtual
# environment, run the following and respond to the prompts.
manage-commcare-cloud configure
```

If you opted out of the final `manage-commcare-cloud configure` step and you have a local environments directory or cloned the repo somewhere other than `~/commcare-cloud` you should set one or both of the following in your bash profile (`~/.profile`) as needed:

```
# for non-standard commcare-cloud repo location
export COMMCARE_CLOUD_REPO=/path/to/your/commcare-cloud

# for local environments (other than $COMMCARE_CLOUD_REPO/environments)
export COMMCARE_CLOUD_ENVIRONMENTS=/path/to/your/environments
```

git-hook setup

After completing the manual setup, make sure you install the git hooks. From the ~/commcare-cloud directory, run the following:

```
(cchq)$ ./git-hooks/install.sh
```

This will make sure you never commit an unencrypted vault.yml file.

Point to your environments directory

commcare-cloud needs to know where environments config directory is located to be able to run commands against the servers in that environment. See *Configuring your CommCare Cloud Environments Directory* to understand what this directory is. The instructions on how this directory is created are part of the CommCare HQ installation docs in *Quick Install on Single Server* and *Install Using Commcare-Cloud on one or more machines*.

Once you have installed commcare-cloud, you can do below to point commcare-cloud to your environments directory.

- Download the environments directory to any path that you own. Make sure the ownership permissions are set right.
- Run `COMMCARE_CLOUD_ENVIRONMENTS=/path/to/environments/folder manage-commcare-cloud configure`.

9.1.2 Configuring your CommCare Cloud Environments Directory

Table of Contents

- *Configuring your CommCare Cloud Environments Directory*
  - *Creating environments directory*

A core component getting commcare-cloud to work to manage your cluster or clusters is the environments directory. This directory contains everything that is different about your organization (authorized users) and cluster environments (IP addresses, machine roles, passwords, optional settings, etc.).

Creating environments directory

This directory is to be created manually when following *Install Using Commcare-Cloud on one or more machines*. When following *Quick Install on Single Server* the script automatically creates this directory. Once created, we recommend you to to manage this via a version control system such as git, so that you can keep track of the changes and share the directory with other team members so that they can perform server administration using commcare-cloud.

Going off the Dimagi example

To make things easy, the real environments dir that Dimagi uses to manage its own environments is committed to the commcare-cloud repo at https://github.com/dimagi/commcare-cloud/tree/master/environments. The easiest way to create a new environment is to model it after one of those.

Layout of an `environments` directory

Your environments dir (traditionally named `environments`) should look like this

- `environments`
  - `_authorized_keys`
  - `_users`
  - `<env1>`
  - `<env2>`

where `<env1>` (and `<env2>`, etc.) are is the environment's name. Here we will describe the contents of directories prefixed with an underscore (_). In the next section we will describe what goes in each environment's directory.

`_authorized_keys`

Each team member who will be granted access to the machines of *any* of the environments should have their public ssh key placed in this directory in a file named `<username>.pub`, where `<username>` is their username as it should appear on each machine. These keys will be used to set up passwordless login to the machines on each cluster they have access to.

For a guide to generating and using ssh keys, see this article on Generating a new SSH key and adding it to the ssh-agent.

_users

Minimally, this directory should contain one file named <your-organization>.yml. If you have more than one environment, *and* you have two environments that require access from a different group of users, you may have one file representing each of these groups. You may name the file anything you wish; you will reference these files by name in the meta.yml file of each environment.

In the sample environments, this file is called admins.yml. You may use and edit this file if you wish.

Each of these files should contain YAML of the following format:

```
dev_users:
  present:
    - <username1>
    - <username2>
    ...
  absent:
    - <username3>
    - <username4>
    ...
```

The **present** section will have a list of users who have access to your servers. The name you add here should be their desired system username, and should correspond to the name of their public key in <username>.pub under ``*_authorized_keys*` <#_authorized_keys>`_.

Each <username> must correspond to that used in a <username>.pub under .

The **absent** section lists those users whose access you want to remove from your servers when running the user update scripts.

If you change this file, you will need to run the ``*update-users*` command <../commands/index.md#update-users>`_

Contents of an environment configuration directory

As mentioned above, commcare-cloud supports servicing multiple cluster environments. Each environment is given a name. For example, at dimagi, our environments are named production, staging, and a few others. This name is used for as the name of the directory, given as <env1>, <env2>, etc. above.

A commcare-cloud environment configuration is made up of the following files:

- ``*app-processes.yml*` <#app-processesyml>`_
- ``*fab-settings.yml*` <#fab-settingsyml>`_
- ``*inventory.ini*` <#inventoryini>`_
- ``*known_hosts*` <#known_hosts>`_
- ``*meta.yml*` <#metayml>`_
- ``*postgresql.yml*` <#postgresqlyml>`_
- ``*proxy.yml*` <#proxyyml>`_
- ``*public.yml*` <#publicyml>`_
- ``*vault.yml*` <#vaultyml>`_

The purpose of each of these files and their formats will be discussed in detail in the following sections.

app-processes.yml

This file determines which background CommCare processes will get run on which machines in the cluster. The file is split into 3 sections each with the same basic format:

```
<section>:
  <host>:
    <process / queue>:
      # process configuration
```

The three sections are as follows:

- `management_commands`: These are usually a single process per cluster and are used to manage various system queues.
- `celery_processes`: Each of the items listed here is a Celery queue.
- `pillows`: Each item listed is a the name of an ETL processor (aka pillow)

Each `<host>` must be a host string.

See app_processes.py for complete list of top-level properties for this file. These are subject to the defaults provided in environmental-defaults/app-processes.yml.

Management Commands

```
management_commands:
  <host>:
    <command-name>:
  <host>:
    ...
  ...
```

Each `<command-name>` must be one of the following:

- `run_submission_reprocessing_queue`: Reprocess failed form submissions
- `queue_schedule_instances`: Populates the SMS queue with scheduled messages
- `handle_survey_actions`: Handles SMS survey actions
- `run_sms_queue`: Processes queued SMS messages
- `run_pillow_retry_queue`: Retry queue for change feed errors

There is no per-process configuration.

Celery Processes

```
celery_processes:
  <host>:
    <queue-name>:
      pooling: [gevent|prefork]  # default prefork
      concurrency: <int>  # Required
      max_tasks_per_child: <int>
  <host>:
```

(continues on next page)

(continued from previous page)

```
    ...
  ...
```

Each <queue-name> must be one of the following values: async_restore_queue, background_queue, case_rule_queue, celery, email_queue, export_download_queue, icds_dashboard_reports_queue, linked_domain_queue, reminder_case_update_queue, reminder_queue, reminder_rule_queue, repeat_record_queue, saved_exports_queue, sumologic_logs_queue, send_report_throttled, sms_queue, submission_reprocessing_queue, ucr_indicator_queue, ucr_queue. For all features to work, each of these queues must appear at least once, and up to once per host.

Under each <queue-name> goes the following parameters:

- concurrency: Required; the concurrency configured on each worker
- pooling: default prefork; specify prefork or gevent for the process pool type used on each worker in this section
- max_tasks_per_child: default 50; only applicable for prefork pooling (corresponds to maxtasksperchild celery worker command line arg)
- num_workers: default 1; the number of workers to create consuming from this queue on this host

The special queue names flower, beat can appear *only* once. These queues take no parameters (can leave as simply {}).

Pillows

```
pillows:
  <host>:
    <ETL-processor-name>:
      num_processes: <int>
  <host>:
    ...
  ...
```

Each *<ETL-processor-name>* must be correspond to the *name* fields specified in *settings.PILLOWTOPS*:

AppDbChangeFeedPillow, ApplicationToElasticsearchPillow, CacheInvalidatePillow, case-pillow, case_messaging_sync_pillow, CaseSearchToElasticsearchPillow, CaseToElasticsearchPillow, DefaultChangeFeedPillow, DomainDbKafkaPillow, FormSubmissionMetadataTrackerPillow, group-pillow, GroupPillow, GroupToUserPillow, kafka-ucr-main, kafka-ucr-static, KafkaDomainPillow, LedgerToElasticsearchPillow, location-ucr-pillow, SqlSMSPillow, UnknownUsersPillow, UpdateUserSyncHistoryPillow, user-pillow, UserCacheInvalidatePillow, UserGroupsDbKafkaPillow, UserPillow, xform-pillow, XFormToElasticsearchPillow,

For all features to work, each of these ETL processors (called "pillows" internally to the CommCare HQ code base, for no good reason beyond historical accident) just listed must appear at least once, and up to once per host. An ETL processor not mentioned will not be run at all.

fab-settings.yml

This file contains basic settings relevent to deploying updated versions CommCare HQ code.

inventory.ini

This is the Ansible Inventory file. It lists all the hosts releveant to the system and provides host groups for the different services. This file can also contain host specific variables like `hostname` or configuration for the encrypted drive.

known_hosts

This file is optional and is auto-generated by running

```
commcare-cloud <env> update-local-known-hosts
```

For commcare-cloud commands that require opening ssh connections, this file is used instead of `~/.ssh/known_hosts` where possible. This allows a team to share a `known_hosts` file that is environment specific, which has both security (depending on how used) and practical benefits (each team member does not have to ssh into each machine and respond `yes` to typical ssh prompt asking whether to trust a given host based on its fingerprint).

meta.yml

This file contains some global settings for the environment.

postgresql.yml

This file contains configuration related to postgresql. For more detail see *Configurating postgresql.yml*.

proxy.yml

This file contains settings related to the Nginx proxy.

public.yml

This file contains the remainder of the settings for the environement that aren't specified in any of the aforementioned files.

vault.yml

This file contains sensitive information such as database passwords. The file is encrypted using Ansible Vault. For information on managing this file see Managing Secrets with Vault

9.1.3 Configurating postgresql.yml

Table of Contents

- *Configurating postgresql.yml*
  - *dbs*
    - *main*
    - *formplayer*
    - *ucr*
    - *synclogs*
    - *form_processing*
    - *"db config" type*
  - *override*
  - *SEPARATE_SYNCLOGS_DB*
  - *SEPARATE_FORM_PROCESSING_DBS*
  - *DEFAULT_POSTGRESQL_HOST*
  - *REPORTING_DATABASES*
  - *LOAD_BALANCED_APPS*

For an example `postgresql.yml` file see environments/production/postgresql.yml.

The following properties are permitted in `postgresql.yml`.

You may notice that some of the properties have a **Status**, which can be either "Custom" or "Deprecated". A status of **Custom** means that the property is a back-door for a heavily customized environment, and should not used in a typical environment. A status of **Deprecated** means that the property exists to support legacy environments that have not yet adopted a new standard, and support may be removed for it in the future; a typical new enviornment should not set these properties either.

dbs

Database-level config such as what machine each db is on. All properties rely on a conceptual "db config" type described in more detail below.

main

- Type: *db config*
- Default values: .. code-block:: yaml

  django_alias: default name: commcarehq django_migrate: True

Configuration for the main db, the db that the majority of tables live in.

formplayer

- Type: *db config*
- Default values: .. code-block:: yaml

    django_alias: null name: formplayer

Configuration for the db that formplayer uses (which does not appear in Django settings).

ucr

- Type: *db config*
- Default values: .. code-block:: yaml

    django_alias: ucr name: commcarehq_ucr django_migrate: False

Configuration for the db that UCRs are copied into.

synclogs

- Type: *db config*
- Default values: .. code-block:: yaml

    django_alias: synclogs name: commcarehq_synclogs django_migrate: True

Configuration for the db that synclog tables live in.

form_processing

- Type: see below Configuration for the db that form, case, and related tables live in.

It is broken down into the proxy config and the config for partitions

proxy

- Type: *db config* with defaults:
- Default values: .. code-block:: yaml

    django_alias: proxy name: commcarehq_proxy

Configuration for the proxy db in the partitioned setup.

proxy_standby

- Type: *db config* with defaults:
- Default values: .. code-block:: yaml

    django_alias: proxy_standby name: commcarehq_proxy_standby

Configuration for the proxy db in the partitioned setup which can be used to query the standby partitions.

partitions

- Type: dict of partition name to (augmented) *db config*

Configurations for each of the partitions of the database. These special configs are augmented with the following property. Partition names must be `p1`, `p2`, ..., `p<N>`.

shards

- Type: pair of integers (list with two elements)

Inclusive start and end indices for the shard range. The `shards` property for all `partitions` combined must cover the entire range of available shards, and the ranges must be in ascending order matching the order of the names of the partitions (`p1`, `p2`, ..., `p<N>`).

"db config" type

The core data type used repeatedly in this configuration is a db config, which has the following properties:

django_alias

- Type: string

Alias for the database. Used as the key for the entry in Django ``*DATABASES*` <https://docs.djangoproject.com/en/2.0/ref/settings/#databases>`_ setting. Most aliases are preset, but for custom databases this can be specified. If missing, the database will not appear in Django settings.

name

- Type: string

Name of the postgresql database. (See ``*NAME*` <https://docs.djangoproject.com/en/2.0/ref/settings/#name>`_.)

host

- Type: host string

The host machine on which this database should live. (See ``*HOST*` <https://docs.djangoproject.com/en/2.0/ref/settings/#host>`_.)

pgbouncer_host

- Type: host string

The host to use to run pgbouncer for this db. Defaults to "host". Cannot be used if the more granular options `pgbouncer_hosts` and `pgbouncer_endpoints` are set.

pgbouncer_hosts

- Type: List of host string

The list of hosts to use to run pgbouncer for this db. Defaults to `[pgbouncer_host]`, and cannot be set explicitly if `pgbouncer_host` is set. If set explicitly, `pgbouncer_endpoint` must also be set.

pgbouncer_endpoint

- Type: host string

The endpoint that other processes should use to communicate with pgbouncer for this db. Defaults to `pgbouncer_host`, and connot be explicitly set if `pgbouncer_host` is set. If set explicitly, `pgbouncer_hosts` must also be set.

The difference between `pgbouncer_endpoints` and `pgbouncer_hosts` is that `pgbouncer_hosts` says where pgbouncer should be installed and running for this db, whereas `pgbouncer_endpoints` says where other machines that want to talk to pgbouncer for this db should point to. Often these are the same machines in which case you can use `pgbouncer_host` as a shortcut to set both.

Some examples where you would want to set the `pgbouncer_endpoints` and `pgbouncer_hosts` independently:

- You have multiple `pgbouncer_hosts` in a network load balancer whose address `pgbouncer_enpoint` is set to.
- You want `pgbouncer1` to be ready to switch over to in case `pgbouncer0` fails. In that case, you set `pgbouncer_hosts` to `[pgbouncer0, pgbouncer1]` and `pgbouncer_endpoint` to `pgbouncer0`.

port

- Type: int

The port to use when connecting to the database. (See ``*PORT*` <https://docs.djangoproject.com/en/2.0/ref/settings/#port>`_.)

user

- Type: string

The username to use when connecting to the database. (See ``*USER*` <https://docs.djangoproject.com/en/2.0/ref/settings/#user>`_.)

password

- Type: string

The password to use when connecting to the database. (See ``*PASSWORD*` <https://docs.djangoproject.com/en/2.0/ref/settings/#password>`_.)

options

- Type: dict

(See ``*OPTIONS*` <https://docs.djangoproject.com/en/2.0/ref/settings/#std:setting-OPTIONS>`_.)

django_migrate

- Type: bool

Whether migrations should be run on this database. For all except in `custom`, this property is automatically determined.

query_stats

- Type: bool
- Default: `False`

Whether query statistics should be collected on this PostgreSQL db using the pg_stat_statements extension.

create

- Type: bool
- Default: `True`

Whether commcare-cloud should create this db (via Ansible).

override

- Type: dict (variables names to values)

Ansible `postgresql` role variables to override. See ansible/roles/postgresql/defaults/main.yml for the complete list of variables.

As with any ansible variable, to override these on a per-host basis, you may set these as inventory host or group variables in `inventory.ini`. Note, however, that variables you set there will not receive any validation, whereas variables set here will be validated against the type in the defaults file linked above.

SEPARATE_SYNCLOGS_DB

- Type: boolean
- Default: `True`
- Status: Deprecated

Whether to save synclogs to a separate postgresql db. A value of `False` may lose support in the near future and is not recommended.

SEPARATE_FORM_PROCESSING_DBS

- Type: boolean
- Default: `True`
- Status: Deprecated

Whether to save form, cases, and related data in a separate set of partitioned postgresql dbs. A value of `False` may lose support in the near future and is not recommended.

DEFAULT_POSTGRESQL_HOST

- Type: host string
- Default: The first machine in the `postgresql` inventory group.

This value will be used as the host for any database without a different host explicitly set in ``*dbs*` <#dbs>`_.

REPORTING_DATABASES

- Type: dict
- Default: `{"ucr": "ucr"}`
- Status: Custom

Specify a mapping of UCR "engines".

The keys define engine aliases, and can be anything. The values are either

- the `django_alias` of a postgreql database

or

- a spec for which (single) database to write to and a weighted list of databases to read from.

The latter option is formatted as follows:

```
WRITE: <django_alias>
READ:
  - [<django_alias>, <weight>]
  - [<django_alias>, <weight>]
  ...
```

where `<weight>` is a low-ish integer. The probability of hitting a given database with weight $W_n$is its normalized weight, i.e. W_n/ (W:sub:1 + … + W_n\ ).

LOAD_BALANCED_APPS

- Type: dict
- Default: {}
- Status: Custom

Specify a list of django apps that can be read from multiple dbs.

The keys are the django app label. The values are a weighted list of databases to read from.

This is formatted as:

```
<app_name>:
  - [<django_alias>, <weight>]
  - [<django_alias>, <weight>]
  ...
```

where <weight> is a low-ish integer. The probability of hitting a given database with weight $W_n$is its normalized weight, i.e. W_n/ (W:sub:1 + … + W_n).

9.1.4 Commands

This page explains how to run commcare-cloud commands to perform various actions on your environment and list of all commcare-cloud commands and their usage.

Running Commands with commcare-cloud

To run any commcare-cloud command you need to install commcare-cloud first (Refer to the installation docs) and activate its virtual environment.

All commcare-cloud commands take the following form:

```
commcare-cloud [--control] [--control-setup {yes,no}] <env> <command> ...
```

Positional Arguments

<env>

server environment to run against

Options

--control

Run command remotely on the control machine.

You can add --control *directly after* commcare-cloud to any command in order to run the command not from the local machine using the local code, but from from the control machine for that environment, using the latest version of commcare-cloud available.

It works by issuing a command to ssh into the control machine, update the code, and run the same command entered locally but with --control removed. For long-running commands, you will have to remain connected to the the control machine for the entirety of the run.

`--control-setup {yes,no}`

Implies –control, and overrides the command's run_setup_on_control_by_default value.

If set to 'yes', the latest version of the branch will be pulled and commcare-cloud will have all its dependencies updated before the command is run. If set to 'no', the command will be run on whatever checkout/install of commcare-cloud is already on the control machine. This defaults to 'yes' if command.run_setup_on_control_by_default is True, otherwise to 'no'.

`cchq` alias

Additionally, `commcare-cloud` is aliased to the easier-to-type `cchq` (short for "CommCare HQ"), so any command you see here can also be run as

```
cchq <env> <command> <args...>
```

Underlying tools and common arguments

The `commcare-cloud` command line tool is by and large a relatively thin wrapper around the other tools it uses: `ansible`, `ansible-playbook`, `ssh`, etc. For every command you run using `commcare-cloud`, it will print out the underlying command that it is running, a faint blue / cyan color. In each case, if you copy and paste the printed command directly, it will have essentially the same affect. (Note too that some commands run multiple underlying commands in sequence, and that each command will be printed.)

Where possible, `commcare-cloud` is set up to pass any unknown arguments to the underlying tool. In addition, there are a number of common arguments that are recognized by many `commcare-cloud` commands, and have similar behavior on across them. Rather than include these on every command they apply to, we will list upfront these common arguments and when they can be used.

To verify availability on any given command, you can always run the command with `-h`.

Ansible-backed commands

For most ansible-backed commands `commcare-cloud` will run in check mode first, and then ask you to confirm before applying the changes. Since check mode does not make sense for all commands, there are some that do not follow this pattern and apply the changes directly.

`--skip-check`

When this argument is included, the "check, ask, apply" behavior described above is circumvented, and the command is instead applied directly

--quiet

Run the command without every prompting for permission to continue. At each point, the affirmative response is assumed.

--branch <branch>

In the specific case that `commcare-cloud` has been installed from git source in egg mode (i.e. using `pip install -e .`), it will always check that the checked-out git branch matches the <branch> that is thus passed in. If this arg is not specified, it defaults to `master`. As a consequence, when running from git branch `master`, there is no need to use the `--branch` arg explicitly.

--output [actionable|minimal]

The callback plugin to use for generating output. See ansible-doc -t callback -l and ansible-doc -t callback.

List of Commands

Internal Housekeeping for your `commcare-cloud` environments

`validate-environment-settings` Command

Validate your environment's configuration files

```
commcare-cloud <env> validate-environment-settings
```

As you make changes to your environment files, you can use this command to check for validation errors or incompatibilities.

`update-local-known-hosts` Command

Update the local known_hosts file of the environment configuration.

```
commcare-cloud <env> update-local-known-hosts
```

You can run this on a regular basis to avoid having to `yes` through the ssh prompts. Note that when you run this, you are implicitly trusting that at the moment you run it, there is no man-in-the-middle attack going on, the type of security breach that the SSH prompt is meant to mitigate against in the first place.

Ad-hoc

`lookup` Command

Lookup remote hostname or IP address

```
commcare-cloud <env> lookup [server]
```

Positional Arguments

`server`

Server name/group: postgresql, proxy, webworkers, … The server name/group may be prefixed with 'username@' to login as a specific user and may be terminated with '[]' to choose one of multiple servers if there is more than one in the group. For example: webworkers[0] will pick the first webworker. May also be omitted for environments with only a single server.

Use '-' for default (django_manage[0])

`ssh` Command

Connect to a remote host with ssh.

```
commcare-cloud <env> ssh [--quiet] [server]
```

This will also automatically add the ssh argument `-A` when `<server>` is `control`.

All trailing arguments are passed directly to `ssh`.

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

Positional Arguments

`server`

Server name/group: postgresql, proxy, webworkers, … The server name/group may be prefixed with 'username@' to login as a specific user and may be terminated with '[]' to choose one of multiple servers if there is more than one in the group. For example: webworkers[0] will pick the first webworker. May also be omitted for environments with only a single server.

Use '-' for default (django_manage[0])

Options

--quiet

Don't output the command to be run.

---

audit-environment Command

This command gathers information about your current environment's state.

```
commcare-cloud <env> audit-environment [--use-factory-auth]
```

State information is saved in the '~/.commcare-cloud/audits' directory. It is a good idea to run this before making any major changes to your environment, as it allows you to have a record of your environment's current state.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

scp Command

Copy file(s) over SSH.

```
commcare-cloud <env> scp [--quiet] source target
```

If a remote host is not specified in either the `source` or `target`, the `source` host defaults to `django_manage[0]`.

Examples:

Copy remote `django_manage` file to local current directory

```
cchq <env> scp /tmp/file.txt .
```

Copy remote .txt files to local /texts/ directory

```
cchq <env> scp webworkers[0]:'/tmp/*.txt' /texts/
```

Copy local file to remote path

```
cchq <env> scp file.txt control:/tmp/other.txt
```

Limitations:

- Multiple `source` arguments are not supported.
- File paths do not auto-complete.
- Unlike normal `scp`, options with values are most easily passed after the `target` argument.

- `scp://` URIs are not supported.
- Copy from remote to remote is not supported.
- Probably many more.

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

Positional Arguments

`source`

Local pathname or remote host with optional path in the form [user@]host:[path].

`target`

Local pathname or remote host with optional path in the form [user@]host:[path].

Options

`--quiet`

Don't output the command to be run.

`run-module` Command

Run an arbitrary Ansible module.

```
commcare-cloud <env> run-module [--use-factory-auth] inventory_group module module_args
```

Example

To print out the `inventory_hostname` ansible variable for each machine, run

```
commcare-cloud <env> run-module all debug "msg={{ '{{' }} inventory_hostname }}"
```

Positional Arguments

`inventory_group`

Machines to run on. Is anything that could be used in as a value for `hosts` in an playbook "play", e.g. `all` for all machines, `webworkers` for a single group, `celery:pillowtop` for multiple groups, etc. See the description in this blog for more detail in what can go here.

module

The name of the ansible module to run. Complete list of built-in modules can be found at Module Index.

module_args

Args for the module, formatted as a single string. (Tip: put quotes around it, as it will likely contain spaces.) Both `arg1=value1 arg2=value2` syntax and `{"arg1": "value1", "arg2": "value2"}` syntax are accepted.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

The ansible options below are available as well

```
  --list-hosts          outputs a list of matching hosts; does not execute
                        anything else
  --playbook-dir BASEDIR
                        Since this tool does not use playbooks, use this as a
                        substitute playbook directory.This sets the relative
                        path for many features including roles/ group_vars/
                        etc.
  --syntax-check        perform a syntax check on the playbook, but do not
                        execute it
  --task-timeout TASK_TIMEOUT
                        set task timeout limit in seconds, must be positive
                        integer.
  --vault-id VAULT_IDS  the vault identity to use
  --version             show program's version number, config file location,
                        configured module search path, module location,
                        executable location and exit
  -B SECONDS, --background SECONDS
                        run asynchronously, failing after X seconds
                        (default=N/A)
  -M MODULE_PATH, --module-path MODULE_PATH
                        prepend colon-separated path(s) to module library (def
                        ault=~/.ansible/plugins/modules:/usr/share/ansible/plu
                        gins/modules)
  -P POLL_INTERVAL, --poll POLL_INTERVAL
                        set the poll interval if using -B (default=15)
  -e EXTRA_VARS, --extra-vars EXTRA_VARS
                        set additional variables as key=value or YAML/JSON, if
                        filename prepend with @
  -f FORKS, --forks FORKS
                        specify number of parallel processes to use
                        (default=50)
  -l SUBSET, --limit SUBSET
```

(continues on next page)

(continued from previous page)

```
                        further limit selected hosts to an additional pattern
  -o, --one-line        condense output
  -t TREE, --tree TREE  log output to this directory
  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
```

Privilege Escalation Options

```
  control how and which user you become as on target hosts

  --become-method BECOME_METHOD
                        privilege escalation method to use (default=sudo), use
                        `ansible-doc -t become -l` to list valid choices.
  -K, --ask-become-pass
                        ask for privilege escalation password
```

Connection Options

```
  control as whom and how to connect to hosts

  --private-key PRIVATE_KEY_FILE, --key-file PRIVATE_KEY_FILE
                        use this file to authenticate the connection
  --scp-extra-args SCP_EXTRA_ARGS
                        specify extra arguments to pass to scp only (e.g. -l)
  --sftp-extra-args SFTP_EXTRA_ARGS
                        specify extra arguments to pass to sftp only (e.g. -f,
                        -l)
  --ssh-common-args SSH_COMMON_ARGS
                        specify common arguments to pass to sftp/scp/ssh (e.g.
                        ProxyCommand)
  --ssh-extra-args SSH_EXTRA_ARGS
                        specify extra arguments to pass to ssh only (e.g. -R)
  -T TIMEOUT, --timeout TIMEOUT
                        override the connection timeout in seconds
                        (default=30)
  -c CONNECTION, --connection CONNECTION
                        connection type to use (default=smart)
  -k, --ask-pass        ask for connection password
  -u REMOTE_USER, --user REMOTE_USER
                        connect as this user (default=None)

Some actions do not make sense in Ad-Hoc (include, meta, etc)
```

`run-shell-command` Command

Run an arbitrary command via the Ansible shell module.

```
commcare-cloud <env> run-shell-command [--silence-warnings] [--use-factory-auth]␣
↪inventory_group shell_command
```

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

Example

```
commcare-cloud <env> run-shell-command all 'df -h | grep /opt/data'
```

to get disk usage stats for `/opt/data` on every machine.

Positional Arguments

`inventory_group`

Machines to run on. Is anything that could be used in as a value for `hosts` in an playbook "play", e.g. `all` for all machines, `webworkers` for a single group, `celery:pillowtop` for multiple groups, etc. See the description in this blog for more detail in what can go here.

`shell_command`

Command to run remotely. (Tip: put quotes around it, as it will likely contain spaces.) Cannot being with `sudo`; to do that use the ansible `--become` option.

Options

`--silence-warnings`

Silence shell warnings (such as to use another module instead).

`--use-factory-auth`

authenticate using the pem file (or prompt for root password if there is no pem file)

The ansible options below are available as well

```
--list-hosts          outputs a list of matching hosts; does not execute
                      anything else
--playbook-dir BASEDIR
                      Since this tool does not use playbooks, use this as a
                      substitute playbook directory.This sets the relative
                      path for many features including roles/ group_vars/
                      etc.
--syntax-check        perform a syntax check on the playbook, but do not
                      execute it
--task-timeout TASK_TIMEOUT
                      set task timeout limit in seconds, must be positive
                      integer.
--vault-id VAULT_IDS  the vault identity to use
--version             show program's version number, config file location,
                      configured module search path, module location,
                      executable location and exit
-B SECONDS, --background SECONDS
                      run asynchronously, failing after X seconds
                      (default=N/A)
-M MODULE_PATH, --module-path MODULE_PATH
                      prepend colon-separated path(s) to module library (def
                      ault=~/.ansible/plugins/modules:/usr/share/ansible/plu
                      gins/modules)
-P POLL_INTERVAL, --poll POLL_INTERVAL
                      set the poll interval if using -B (default=15)
-e EXTRA_VARS, --extra-vars EXTRA_VARS
                      set additional variables as key=value or YAML/JSON, if
                      filename prepend with @
-f FORKS, --forks FORKS
                      specify number of parallel processes to use
                      (default=50)
-l SUBSET, --limit SUBSET
                      further limit selected hosts to an additional pattern
-o, --one-line        condense output
-t TREE, --tree TREE  log output to this directory
-v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                      connection debugging)
```

Privilege Escalation Options

```
control how and which user you become as on target hosts

--become-method BECOME_METHOD
                      privilege escalation method to use (default=sudo), use
                      `ansible-doc -t become -l` to list valid choices.
-K, --ask-become-pass
                      ask for privilege escalation password
```

Connection Options

```
  control as whom and how to connect to hosts

  --private-key PRIVATE_KEY_FILE, --key-file PRIVATE_KEY_FILE
                        use this file to authenticate the connection
  --scp-extra-args SCP_EXTRA_ARGS
                        specify extra arguments to pass to scp only (e.g. -l)
  --sftp-extra-args SFTP_EXTRA_ARGS
                        specify extra arguments to pass to sftp only (e.g. -f,
                        -l)
  --ssh-common-args SSH_COMMON_ARGS
                        specify common arguments to pass to sftp/scp/ssh (e.g.
                        ProxyCommand)
  --ssh-extra-args SSH_EXTRA_ARGS
                        specify extra arguments to pass to ssh only (e.g. -R)
  -T TIMEOUT, --timeout TIMEOUT
                        override the connection timeout in seconds
                        (default=30)
  -c CONNECTION, --connection CONNECTION
                        connection type to use (default=smart)
  -k, --ask-pass        ask for connection password
  -u REMOTE_USER, --user REMOTE_USER
                        connect as this user (default=None)

Some actions do not make sense in Ad-Hoc (include, meta, etc)
```

`send-datadog-event` Command

Track an infrastructure maintainance event in Datadog

```
commcare-cloud <env> send-datadog-event [--tags [TAGS ...]] [--alert_type {error,warning,
↪info,success}]
                                        event_title event_text
```

Positional Arguments

`event_title`

Title of the datadog event.

event_text

Text content of the datadog event.

Options

--tags [TAGS ...]

Additional tags e.g. host:web2

--alert_type {error,warning,info,success}

Alert type.

---

django-manage Command

Run a django management command.

```
commcare-cloud <env> django-manage [--tmux] [--server SERVER] [--release RELEASE] [--tee␣
↪TEE_FILE] [--quiet]
```

`commcare-cloud <env> django-manage ...` runs `./manage.py ...` on the first django_manage machine of <env> or server you specify. Omit <command> to see a full list of possible commands.

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

Example

To open a django shell in a tmux window using the `2018-04-13_18.16` release.

```
commcare-cloud <env> django-manage --tmux --release 2018-04-13_18.16 shell
```

To do this on a specific server

```
commcare-cloud <env> django-manage --tmux shell --server web0
```

Options

--tmux

If this option is included, the management command will be run in a new tmux window under the `cchq` user. You may then exit using the customary tmux command `^b d`, and resume the session later. This is especially useful for long-running commands.

The tmux session will be unique to your user. If you want to be able to share your session with other users, create the tmux session manually on the machine under a shared user account.

--server SERVER

Server to run management command on. Defaults to first server under django_manage inventory group

--release RELEASE

Name of release to run under. E.g. '2018-04-13_18.16'. If none is specified, the `current` release will be used.

--tee TEE_FILE

Tee output to the screen and to this file on the remote machine

--quiet

Don't output the command to be run.

`tmux` Command

Connect to a remote host with ssh and open a tmux session.

```
commcare-cloud <env> tmux [--quiet] [server] [remote_command]
```

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

Example

Rejoin last open tmux window.

```
commcare-cloud <env> tmux -
```

Positional Arguments

server

Server name/group: postgresql, proxy, webworkers, … The server name/group may be prefixed with 'username@' to login as a specific user and may be terminated with '[]' to choose one of multiple servers if there is more than one in the group. For example: webworkers[0] will pick the first webworker. May also be omitted for environments with only a single server.

Use '-' for default (django_manage[0])

remote_command

Command to run in the tmux. If a command is specified, then it will always run in a new window. If a command is *not* specified, then it will rejoin the most recently visited tmux window; only if there are no currently open tmux windows will a new one be opened.

Options

--quiet

Don't output the command to be run.

---

export-sentry-events Command

Export Sentry events. One line per event JSON.

```
commcare-cloud <env> export-sentry-events -k API_KEY -i ISSUE_ID [--full] [--cursor␣
↪CURSOR]
```

Options

-k API_KEY, --api-key API_KEY

Sentry API Key

-i ISSUE_ID, --issue-id ISSUE_ID

Sentry project ID

--full

Export the full event details

--cursor CURSOR

Starting position for the cursor

`pillow-topic-assignments` Command

Print out the list of Kafka partitions assigned to each pillow process.

```
commcare-cloud <env> pillow-topic-assignments [--csv] pillow_name
```

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

Positional Arguments

`pillow_name`

Name of the pillow.

Options

`--csv`

Output as CSV

Operational

`secrets` Command

View and edit secrets through the CLI

```
commcare-cloud <env> secrets {view,edit,list-append,list-remove} secret_name
```

Positional Arguments

`{view,edit,list-append,list-remove}`

`secret_name`

`migrate-secrets` Command

Migrate secrets from one backend to another

```
commcare-cloud <env> migrate-secrets [--to-backend TO_BACKEND] from_backend
```

Positional Arguments

from_backend

Options

--to-backend TO_BACKEND

---

ping Command

Ping specified or all machines to see if they have been provisioned yet.

```
commcare-cloud <env> ping [--use-factory-auth] inventory_group
```

Positional Arguments

inventory_group

Machines to run on. Is anything that could be used in as a value for `hosts` in an playbook "play", e.g. `all` for all machines, `webworkers` for a single group, `celery:pillowtop` for multiple groups, etc. See the description in this blog for more detail in what can go here.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

ansible-playbook Command

(Alias `ap`)

Run a playbook as you would with ansible-playbook

```
commcare-cloud <env> ansible-playbook [--use-factory-auth] playbook
```

By default, you will see –check output and then asked whether to apply.

Example

```
commcare-cloud <env> ansible-playbook deploy_proxy.yml --limit=proxy
```

Positional Arguments

playbook

The ansible playbook .yml file to run. Options are the `*.yml` files located under `commcare_cloud/ansible` which is under `src` for an egg install and under `<virtualenv>/lib/python<version>/site-packages` for a wheel install.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

The ansible-playbook options below are available as well

```
  --flush-cache         clear the fact cache for every host in inventory
  --force-handlers      run handlers even if a task fails
  --list-hosts          outputs a list of matching hosts; does not execute
                        anything else
  --list-tags           list all available tags
  --list-tasks          list all tasks that would be executed
  --skip-tags SKIP_TAGS
                        only run plays and tasks whose tags do not match these
                        values
  --start-at-task START_AT_TASK
                        start the playbook at the task matching this name
  --step                one-step-at-a-time: confirm each task before running
  --syntax-check        perform a syntax check on the playbook, but do not
                        execute it
  --vault-id VAULT_IDS  the vault identity to use
  --version             show program's version number, config file location,
                        configured module search path, module location,
                        executable location and exit
  -M MODULE_PATH, --module-path MODULE_PATH
                        prepend colon-separated path(s) to module library (def
                        ault=~/.ansible/plugins/modules:/usr/share/ansible/plu
                        gins/modules)
  -e EXTRA_VARS, --extra-vars EXTRA_VARS
                        set additional variables as key=value or YAML/JSON, if
                        filename prepend with @
  -f FORKS, --forks FORKS
                        specify number of parallel processes to use
                        (default=50)
  -t TAGS, --tags TAGS  only run plays and tasks tagged with these values
```

(continues on next page)

(continued from previous page)

```
  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
```

Connection Options

```
  control as whom and how to connect to hosts

  --private-key PRIVATE_KEY_FILE, --key-file PRIVATE_KEY_FILE
                        use this file to authenticate the connection
  --scp-extra-args SCP_EXTRA_ARGS
                        specify extra arguments to pass to scp only (e.g. -l)
  --sftp-extra-args SFTP_EXTRA_ARGS
                        specify extra arguments to pass to sftp only (e.g. -f,
                        -l)
  --ssh-common-args SSH_COMMON_ARGS
                        specify common arguments to pass to sftp/scp/ssh (e.g.
                        ProxyCommand)
  --ssh-extra-args SSH_EXTRA_ARGS
                        specify extra arguments to pass to ssh only (e.g. -R)
  -T TIMEOUT, --timeout TIMEOUT
                        override the connection timeout in seconds
                        (default=30)
  -c CONNECTION, --connection CONNECTION
                        connection type to use (default=smart)
  -k, --ask-pass        ask for connection password
  -u REMOTE_USER, --user REMOTE_USER
                        connect as this user (default=None)
```

Privilege Escalation Options

```
  control how and which user you become as on target hosts

  --become-method BECOME_METHOD
                        privilege escalation method to use (default=sudo), use
                        `ansible-doc -t become -l` to list valid choices.
  --become-user BECOME_USER
                        run operations as this user (default=root)
  -K, --ask-become-pass
                        ask for privilege escalation password
  -b, --become          run operations with become (does not imply password
                        prompting)
```

`deploy-stack` Command

(Alias `aps`)

Run the ansible playbook for deploying the entire stack.

```
commcare-cloud <env> deploy-stack [--use-factory-auth] [--first-time]
```

Often used in conjunction with –limit and/or –tag for a more specific update.

Options

`--use-factory-auth`

authenticate using the pem file (or prompt for root password if there is no pem file)

`--first-time`

Use this flag for running against a newly-created machine.

It will first use factory auth to set up users, and then will do the rest of deploy-stack normally, but skipping check mode.

Running with this flag is equivalent to

```
commcare-cloud <env> bootstrap-users <...args>
commcare-cloud <env> deploy-stack --skip-check --skip-tags=users <...args>
```

If you run and it fails half way, when you're ready to retry, you're probably better off running

```
commcare-cloud <env> deploy-stack --skip-check --skip-tags=users <...args>
```

since if it made it through bootstrap-users you won't be able to run bootstrap-users again.

`update-config` Command

Run the ansible playbook for updating app config.

```
commcare-cloud <env> update-config
```

This includes django `localsettings.py` and formplayer `application.properties`.

`after-reboot` Command

Bring a just-rebooted machine back into operation.

```
commcare-cloud <env> after-reboot [--use-factory-auth] inventory_group
```

Includes mounting the encrypted drive. This command never runs in check mode.

Positional Arguments

`inventory_group`

Machines to run on. Is anything that could be used in as a value for `hosts` in an playbook "play", e.g. `all` for all machines, `webworkers` for a single group, `celery:pillowtop` for multiple groups, etc. See the description in this blog for more detail in what can go here.

Options

`--use-factory-auth`

authenticate using the pem file (or prompt for root password if there is no pem file)

`bootstrap-users` Command

Add users to a set of new machines as root.

```
commcare-cloud <env> bootstrap-users [--use-factory-auth]
```

This must be done before any other user can log in.

This will set up machines to reject root login and require password-less logins based on the usernames and public keys you have specified in your environment. This can only be run once per machine; if after running it you would like to run it again, you have to use `update-users` below instead.

Options

`--use-factory-auth`

authenticate using the pem file (or prompt for root password if there is no pem file)

---

update-users Command

Bring users up to date with the current CommCare Cloud settings.

```
commcare-cloud <env> update-users [--use-factory-auth]
```

In steady state this command (and not `bootstrap-users`) should be used to keep machine user accounts, permissions, and login information up to date.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

update-user-key Command

Update a single user's public key (because update-users takes forever).

```
commcare-cloud <env> update-user-key [--use-factory-auth] username
```

Positional Arguments

username

username who owns the public key

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

update-supervisor-confs Command

Updates the supervisor configuration files for services required by CommCare.

```
commcare-cloud <env> update-supervisor-confs [--use-factory-auth]
```

These services are defined in app-processes.yml.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

fab Command

Placeholder for obsolete fab commands

```
commcare-cloud <env> fab [-l] [fab_command]
```

Positional Arguments

fab_command

The name of the obsolete fab command.

Options

-l

Use `-l` instead of a command to see the full list of commands.

Obsolete fab commands

```
Obsolete fab command       Replaced by 'commcare-cloud ENV ...'
--------------------       ------------------------------------
check_status               ping all
                           service postgresql status
                           service elasticsearch status

clean_releases             clean-releases [--keep=N]
deploy_commcare            deploy commcare
kill_stale_celery_workers  kill-stale-celery-workers
manage                     django-manage
perform_system_checks      perform-system-checks
preindex_views             preindex-views
restart_services           service commcare restart
restart_webworkers         service webworker restart
rollback                   deploy commcare --resume=PREVIOUS_RELEASE

Use the 'list-releases' command to get valid release names.
```

(continues on next page)

(continued from previous page)

```
rollback_formplayer         ansible-playbook rollback_formplayer.yml --tags=rollback
setup_limited_release       deploy commcare --private [--keep-days=N] [--commcare-rev=HQ_
↪BRANCH]
setup_release               deploy commcare --private --limit=all [--keep-days=N] [--
↪commcare-rev=HQ_BRANCH]
start_celery                service celery start
start_pillows               service pillowtop start
stop_celery                 service celery stop
stop_pillows                service pillowtop stop
supervisorctl               service NAME ACTION
update_current              deploy commcare --resume=RELEASE_NAME
```

`deploy` Command

Deploy CommCare

```
commcare-cloud <env> deploy [--resume RELEASE_NAME] [--private] [-l SUBSET] [--keep-days␣
↪KEEP_DAYS] [--skip-record]
                            [--commcare-rev COMMCARE_REV] [--ignore-kafka-checkpoint-
↪warning] [--update-config]
                            [{commcare,formplayer} ...]
```

Positional Arguments

`{commcare,formplayer}`

Component(s) to deploy. Default is 'commcare', or if always_deploy_formplayer is set in meta.yml, 'commcare formplayer'

Options

`--resume RELEASE_NAME`

Rather than starting a new deploy, resume a previous release. This option can be used to "rollback" to a previous release. Use the 'list-releases' command to get valid release names.

`--private`

Set up a private release for running management commands. This option implies –limit=django_manage. Use –limit=all to set up a private release on all applicable hosts.

-l SUBSET, --limit SUBSET

Limit selected hosts.

--keep-days KEEP_DAYS

The number of days to keep the release before it will be purged.

--skip-record

Skip the steps involved in recording and announcing the fact of the deploy.

--commcare-rev COMMCARE_REV

The name of the commcare-hq git branch, tag, or SHA-1 commit hash to deploy.

--ignore-kafka-checkpoint-warning

Do not block deploy if Kafka checkpoints are unavailable.

--update-config

Generate new localsettings.py rather than copying from the previous release.

---

list-releases Command

List names that can be passed to `deploy --resume=RELEASE_NAME`

```
commcare-cloud <env> list-releases [--limit LIMIT]
```

Options

--limit LIMIT

Run command on limited host(s). Default: webworkers[0]

`clean-releases` Command

Cleans old and failed deploys from the ~/www/ENV/releases/ directory.

```
commcare-cloud <env> clean-releases [-k N] [-x [EXCLUDE ...]]
```

Options

`-k N, --keep N`

The number of releases to retain. Default: 3

`-x [EXCLUDE ...], --exclude [EXCLUDE ...]`

Extra release names to exclude from cleanup, in addition to the automatic exclusions such as the current release.

`preindex-views` Command

```
commcare-cloud <env> preindex-views [--commcare-rev COMMCARE_REV] [--release RELEASE_
↪NAME]
```

Set up a private release on the first pillowtop machine and run preindex_everything with that release.

Options

`--commcare-rev COMMCARE_REV`

The name of the commcare-hq git branch, tag, or SHA-1 commit hash to deploy.

`--release RELEASE_NAME`

Use/resume an existing release rather than creating a new one.

`service` Command

Manage services.

```
commcare-cloud <env> service [--only PROCESS_PATTERN]
                             {celery,citusdb,commcare,couchdb2,elasticsearch,
↪elasticsearch-classic,formplayer,kafka,nginx,pillowtop,postgresql,rabbitmq,redis,
↪webworker}
                             [{celery,citusdb,commcare,couchdb2,elasticsearch,
```

(continues on next page)

(continued from previous page)

```
↪elasticsearch-classic,formplayer,kafka,nginx,pillowtop,postgresql,rabbitmq,redis,
↪webworker} ...]
                              {start,stop,restart,status,logs,help}
```

Example

```
cchq <env> service postgresql status
cchq <env> service celery help
cchq <env> service celery logs
cchq <env> service celery restart --limit <host>
cchq <env> service celery restart --only <queue-name>,<queue-name>:<queue_num>
cchq <env> service pillowtop restart --limit <host> --only <pillow-name>
```

Services are grouped together to form conceptual service groups. Thus the `postgresql` service group applies to both the `postgresql` service and the `pgbouncer` service. We'll call the actual services "subservices" here.

Positional Arguments

{celery,citusdb,commcare,couchdb2,elasticsearch,elasticsearch-classic,formplayer,kafka, nginx,pillowtop,postgresql,rabbitmq,redis,webworker}

The name of the service group(s) to apply the action to. There is a preset list of service groups that are supported. More than one service may be supplied as separate arguments in a row.

{start,stop,restart,status,logs,help}

Action can be `status`, `start`, `stop`, `restart`, or `logs`. This action is applied to every matching service.

Options

--only PROCESS_PATTERN

Sub-service name to limit action to. Format as 'name' or 'name:number'. Use 'help' action to list all options.

---

migrate-couchdb Command

(Alias `migrate_couchdb`)

Perform a CouchDB migration

```
commcare-cloud <env> migrate-couchdb [--no-stop] migration_plan {describe,plan,migrate,
↪commit,clean}
```

This is a recent and advanced addition to the capabilities, and is not yet ready for widespread use. At such a time as it is ready, it will be more thoroughly documented.

Positional Arguments

migration_plan

Path to migration plan file

{describe,plan,migrate,commit,clean}

Action to perform

- describe: Print out cluster info
- plan: generate plan details from migration plan
- migrate: stop nodes and copy shard data according to plan
- commit: update database docs with new shard allocation
- clean: remove shard files from hosts where they aren't needed

Options

--no-stop

When used with migrate, operate on live couchdb cluster without stopping nodes.

This is potentially dangerous. If the sets of a shard's old locations and new locations are disjoint—i.e. if there are no "pivot" locations for a shard—then running migrate and commit without stopping couchdb will result in data loss. If your shard reallocation has a pivot location for each shard, then it's acceptable to do live.

---

downtime Command

Manage downtime for the selected environment.

```
commcare-cloud <env> downtime [-m MESSAGE] [-d DURATION] {start,end}
```

This notifies Datadog of the planned downtime so that is is recorded in the history, and so that during it service alerts are silenced.

Positional Arguments

{start,end}

Options

-m MESSAGE, --message MESSAGE

Optional message to set on Datadog.

-d DURATION, --duration DURATION

Max duration in hours for the Datadog downtime after which it will be auto-cancelled. This is a safeguard against downtime remaining active and preventing future alerts. Default: 24 hours

---

copy-files Command

Copy files from multiple sources to targets.

```
commcare-cloud <env> copy-files plan_path {prepare,copy,cleanup}
```

This is a general purpose command that can be used to copy files between hosts in the cluster.

Files are copied using `rsync` from the target host. This tool assumes that the specified user on the source host has permissions to read the files being copied.

The plan file must be formatted as follows:

```
source_env: env1 (optional if source is different from target;
                  SSH access must be allowed from the target host(s) to source host(s))
copy_files:
  - <target-host>:
      - source_host: <source-host>
        source_user: <user>
        source_dir: <source-dir>
        target_dir: <target-dir>
        rsync_args: []
        files:
          - test/
          - test1/test-file.txt
        exclude:
          - logs/*
          - test/temp.txt
```

- **copy_files**: Multiple target hosts can be listed.
- **target-host**: Hostname or IP of the target host. Multiple source definitions can be listed for each target host.
- **source-host**: Hostname or IP of the source host.
- **source-user**: (optional) User to ssh as from target to source. Defaults to 'ansible'. This user must have permissions to read the files being copied.
- **source-dir**: The base directory from which all source files referenced.
- **target-dir**: Directory on the target host to copy the files to.
- **rsync_args**: Additional arguments to pass to rsync.
- **files**: List of files to copy. File paths are relative to `source-dir`. Directories can be included and must end with a `/`.
- **exclude**: (optional) List of relative paths to exclude from the *source-dir*. Supports wildcards e.g. "logs/*".

Positional Arguments

plan_path

Path to plan file

{prepare,copy,cleanup}

Action to perform

- prepare: generate the scripts and push them to the target servers
- copy: execute the scripts
- cleanup: remove temporary files and remote auth

---

list-postgresql-dbs Command

Example:

```
commcare-cloud <env> list-postgresql-dbs [--compare]
```

To list all database on a particular environment.

```
commcare-cloud <env> list-postgresql-dbs
```

Options

--compare

Gives additional databases on the server.

---

celery-resource-report Command

Report of celery resources by queue.

```
commcare-cloud <env> celery-resource-report [--show-workers] [--csv]
```

Options

`--show-workers`

Includes the list of worker nodes for each queue

`--csv`

Output table as CSV

`pillow-resource-report` Command

Report of pillow resources.

```
commcare-cloud <env> pillow-resource-report [--csv]
```

Options

`--csv`

Output table as CSV

`kill-stale-celery-workers` Command

Kill celery workers that failed to properly go into warm shutdown.

```
commcare-cloud <env> kill-stale-celery-workers
```

When used with –control, this command skips the slow setup. To force setup, use –control-setup=yes instead.

`perform-system-checks` Command

```
commcare-cloud <env> perform-system-checks
```

Check the Django project for potential problems in two phases, first check all apps, then run database checks only.

See https://docs.djangoproject.com/en/dev/ref/django-admin/#check

---

`couchdb-cluster-info` Command

Output information about the CouchDB cluster.

```
commcare-cloud <env> couchdb-cluster-info [--raw] [--shard-counts] [--database DATABASE]␣
↪[--couch-port COUCH_PORT]
                                          [--couch-local-port COUCH_LOCAL_PORT] [--
↪couchdb-version COUCHDB_VERSION]
```

Shard counts are displayed as follows

```
* a single number if all nodes have the same count
* the count on the first node followed by the difference in each following node
  e.g. 2000,+1,-2 indicates that the counts are 2000,2001,1998
```

Options

`--raw`

Output raw shard allocations as YAML instead of printing tables

`--shard-counts`

Include document counts for each shard

`--database DATABASE`

Only show output for this database

`--couch-port COUCH_PORT`

CouchDB port. Defaults to 15984

`--couch-local-port COUCH_LOCAL_PORT`

CouchDB local port (only applicable to CouchDB version < '3.0.0'). Defaults to 15986

--couchdb-version COUCHDB_VERSION

CouchDB version. Assumes '2.3.1' or couchdb_version if set in public.yml

---

terraform Command

Run terraform for this env with the given arguments

```
commcare-cloud <env> terraform [--skip-secrets] [--apply-immediately] [--username␣
↪USERNAME]
```

Options

--skip-secrets

Skip regenerating the secrets file.

Good for not having to enter vault password again.

--apply-immediately

Apply immediately regardless fo the default.

In RDS where the default is to apply in the next maintenance window, use this to apply immediately instead. This may result in a service interruption.

--username USERNAME

The username of the user whose public key will be put on new servers.

Normally this would be *your* username. Defaults to the value of the COMMCARE_CLOUD_DEFAULT_USERNAME environment variable or else the username of the user running the command.

---

terraform-migrate-state Command

Apply unapplied state migrations in commcare_cloud/commands/terraform/migrations

```
commcare-cloud <env> terraform-migrate-state [--replay-from REPLAY_FROM]
```

This migration tool should exist as a generic tool for terraform, but terraform is still not that mature, and it doesn't seem to exist yet.

Terraform assigns each resource an address so that it can map it back to the code. However, often when you change the code, the addresses no longer map to the same place. For this, terraform offers the terraform state mv <address> <new_address> command, so you can tell it how existing resources map to your new code.

This is a tedious task, and often follows a very predictable renaming pattern. This command helps fill this gap.

Options

--replay-from REPLAY_FROM

Set the last applied migration value to this number before running. Will begin running migrations after this number, not including it.

---

aws-sign-in Command

Use your MFA device to "sign in" to AWS for <duration> minutes (default 30)

```
commcare-cloud <env> aws-sign-in [--duration-minutes DURATION_MINUTES]
```

This will store the temporary session credentials in ~/.aws/credentials under a profile named with the pattern "<aws_profile>:profile". After this you can use other AWS-related commands for up to <duration> minutes before having to sign in again.

Options

--duration-minutes DURATION_MINUTES

Stay signed in for this many minutes

---

aws-list Command

List endpoints (ec2, rds, etc.) on AWS

```
commcare-cloud <env> aws-list
```

---

aws-fill-inventory Command

Fill inventory.ini.j2 using AWS resource values cached in aws-resources.yml

```
commcare-cloud <env> aws-fill-inventory [--cached]
```

If –cached is not specified, also refresh aws-resources.yml to match what is actually in AWS.

Options

--cached

Use the values set in aws-resources.yml rather than fetching from AWS.

This runs much more quickly and gives the same result, provided no changes have been made to our actual resources in AWS.

---

openvpn-activate-user Command

Give a OpenVPN user a temporary password (the ansible user password)

```
commcare-cloud <env> openvpn-activate-user [--use-factory-auth] vpn_user
```

to allow the user to connect to the VPN, log in, and change their password using

```
cchq <env> openvpn-claim-user
```

Positional Arguments

vpn_user

The user to activate.

Must be one of the defined ssh users defined for the environment.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

openvpn-claim-user Command

Claim an OpenVPN user as your own, setting its password

```
commcare-cloud <env> openvpn-claim-user [--use-factory-auth] vpn_user
```

Positional Arguments

vpn_user

The user to claim.

Must be one of the defined ssh users defined for the environment.

Options

--use-factory-auth

authenticate using the pem file (or prompt for root password if there is no pem file)

---

forward-port Command

Port forward to access a remote admin console

```
commcare-cloud <env> forward-port {flower,couch,elasticsearch}
```

Positional Arguments

{flower,couch,elasticsearch}

The remote service to port forward. Must be one of couch,elasticsearch,flower.

9.1.5 Slack Notifications from CommCare Cloud

commcare-cloud can be configured to send event notifications to Slack such as pre and post deploy messages.

To configure commcare-cloud to send messages to Slack:

1. Create a Slack app and copy the access token.

   The app will require the following authentication scopes:

   1. channels:join
   2. chat:write
   3. reactions:write

   For an already installed app the app token can be found in the app settings under "Install App".

2. Set the access token as a commcare-cloud secret:

   Run the following command and paste in the token when prompted:

   ```
   commcare-cloud <env> secrets edit slack_token
   ```

3. Set the value of `slack_notifications_channel` in the environment `meta.yml` file. This should be the ID of the Slack channel to send notifications to.

```
slack_notifications_channel: "C0WLJ3XYZ"
```

   To get the ID you can open the channel in a web browser and copy the ID from the URL:

```
https://app.slack.com/client/.../C0WLJ3XYZ
```

9.2 User Access Management

9.2.1 Setting up CommCare HQ Server Administrators

It is possible that a team hosting CommCare HQ may have multiple developers managing the server environment. For a developer (referred to as user in rest of this doc) to be able to enter and execute commands on the server using commcare cloud, users have to be added and then configured. Through the lifetime of a project new users might need to be added to the servers and some users may need to be removed.

This document describes the process of how to set up additional users on the server and how to remove an existing user.

The process to onboard new users as server administrators involves creating an account for the users on all the servers, adding their SSH key to the servers and finally the users setting up their commcare-cloud on their local machine or on a shared server to execute commcare-cloud commands that perform server administration.

Users can be added to the server in one of two ways:

- During installation
- After installation (in steady state)

After the user(s) have been added, the new users need to set up their commcare-cloud. This involves

1. Cloning commcare cloud repo.
2. Installing commcare cloud.
3. Configuring commcare cloud to use the already existing environments folder that is set up during installation.

Adding users during installation

This is done during the installation. The process is described in the installation documentation in *Install Using Commcare-Cloud on one or more machines* and *Quick Install on Single Server*.

Adding and Removing users in steady state

Only users who have access to the servers and the *environments config directory* created during the installation can add new users or remove users.

In order for a user to be added (or removed) to all the servers in the environment in steady state, the following steps are required (any user that has already been successfully added can execute this).

1. Add `<username>.pub` key to `environments/_authorized_keys` folder.
2. Add <username> to the `present` section in the admins.yml file in `environments/_users/admin.yml`.
3. Run `cchq <env_name> update-users` to create the newly added users and add their SSH keys to the servers.

To remove a user, add <username> to the `absent` section in the admins.yml file in `environments/_users/admin.yml` and run the `update-users` command.

Running Commands using commcare-cloud

The new users need to install commcare-cloud and configre commcare-cloud to the the environments config directory to be able to run the commands. Refer to *commcare-cloud reference* to know how to do this.

9.3 Firefighting Production Issues

This section has information on how to debug issues that might occur in your CommCare HQ instance.

This is a firefighting guide to help troubleshoot various issues that might come up while running a CommCare HQ Server.

Note: All Datadog links will be specific and private to Dimagi employees. If datadog releases a feature to share dashboard configurations, we will happily share configurations in this repository.

9.3.1 Firefighting Guide

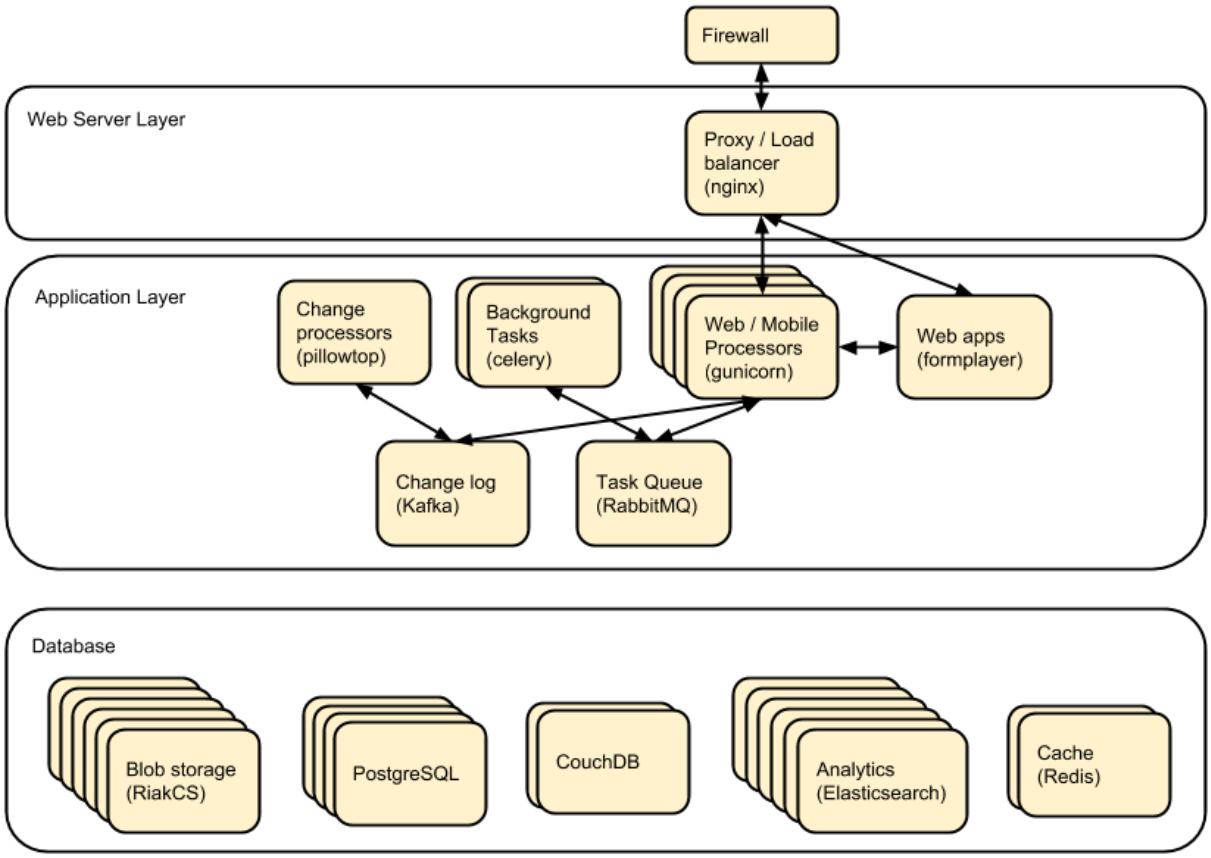
Table of Contents

- *Firefighting Guide*
  - *HQ Architecture and Machines*
  - *High-level System Monitoring and Alerts*
  - *Control machine log files*
  - *In case of a reboot*
  - *In case of network outage*
  - *Service Information*
  - *Switching to Maintenance Page*
  - *Couch 2.0*
  - *Nginx*
  - *NFS & File serving / downloads*
  - *Pgbouncer*
  - *Postgres Troubleshooting*
  - *Celery*
  - *Elasticsearch*
  - *Redis*
  - *Pillows / Pillowtop / Change feed*

- *Formplayer / Cloudcare / Webapps*
- *Full Drives / Out of Disk Space*
- *Network Connection Issues (please help expand)*
- *Tips and Tricks*
- *Some Short Write-ups and Examples*
- *Backups*
- *SMS Gateways*

For a more user-friendly guide check out Cory's brown bag on the topic.

HQ Architecture and Machines



High-level System Monitoring and Alerts

HQ Vitals - Various graphs on datadog

Datadog Alerts) - these are also often reported on #hq-ops or #hq-ops-priority on slack

https://www.commcarehq.org/hq/admin/system/ - catchall system info, contains deploy history, pillowtop info, and a bunch of other stuff

https://www.commcarehq.org/hq/admin/system/check_services - plaintext URL that checks the status of a bunch of support services

Control machine log files

There are two log files on the control machine that might be useful to reference if you need to know what commands were executed. These files are located in the */var/log/* directory and are:

- ansible.log: Shows the output of ansible commands.
- commands.log: Shows the commands that were run and by which user.

In case of a reboot

After reboot, whether or not it was deliberate

In case a machine has automatically rebooted or you needed to reboot a machine, you will need to run the after-reboot protocol directly afterwards. You can specify a single server by IP address, a single server by its name in inventory.ini. You will have to confirm to run, and then provide the vault password for your environment.

```
$ cchq <env> after-reboot [all|<group>|<server>]
Do you want to apply without running the check first? [y/N]y
```

You don't always control the reboot process (sometimes our provider will expectedly or unexpectedly reboot a VM), but if you do, here's the process end to end:

```
# Before shutting down VMs
$ cchq <env> service commcare stop
$ cchq <env> ansible-playbook stop_servers.yml

# After restarting VMs
$ cchq <env> after-reboot all
$ cchq <env> django-manage check_services  # ping various auxiliary services to make␣
↪sure they're up
  # if any services aren't running, you may have to manually start them:
$ # cchq <env> service postgres start
$ cchq <env> service commcare start  # start app processes
```

Applying file system check for every reboot

Checking the file system for errors is an important part of Linux system administration. It is a good troubleshooting step to perform when encountering bad performance on read and write times, or file system errors.

This guide will walk you through the process on how to force **fsck** to perform a file system check on the *next system reboot* or force file system check for any desired number of system reboots on *root mount point*.

1. View and modify PASS value in /etc/fstab

First, use the blkid command to figure out the UUID value of the file system you want to check.

```
$ blkid /dev/sda3
```

output might look like this:

```
/dev/sda3:  UUID="c020d4d8-c104-4140-aafc-24f7f89f8629" BLOCK_SIZE="4096" TYPE="ext4"
PARTUUID="22ada8f4-5222-4049-b0fe-a3274516754d"
```

Then, grep for that UUID in the **/etc/fstab** file.

```
$ grep c020d4d8-c104-4140-aafc-24f7f89f8629 /etc/fstab
```

output:

```
UUID=c020d4d8-c104-4140-aafc-24f7f89f8629 / ext4 defaults 0 0
```

The last column that is a column 6, aka **fsck PASS column** is used by fsck to determine whether fsck should check filesystem before it is mounted and in which order given partitions in /etc/fstab should be checked. Possible entries for fstab PASS column are 0,1 and 2.

0 - disabled, that is do not check the filesystem.

1 - partition with this PASS value has a higher priority and is checked first. This value is usually set to root/partition.

2 - partitions with this PASS value will be checked last

```
change the value of last column to 1 and save exit
```

```
eg. UUID=c020d4d8-c104-4140-aafc-24f7f89f8629 / ext4 defaults 0 1
```

Note: the above setting will apply a filesystem check on the root mount `/`

2. Change "Maximum number of mounts"

To ensure that your file system is checked on the next reboot, we need to manipulate the filesystem's "Maximum mount count" parameter. The following **tune2fs** command will ensure that filesystem /dev/sdX is checked every time your Linux system reboots. Please note that for this to happen the fsck's PASS value in /etc/fstab must be set to a positive integer as discussed above.

```
sudo tune2fs -c 1 /dev/sdX
```

Note: /dev/sdX device where `/` is mounted

3. Change kernel parameter

To try to fix potential problems without getting any prompts, pass the -y option to fsck.

```
eg. sudo fsck -y /dev/sda2
```

This way, you say **yes, try to fix all detected errors** without being prompted every time. If no errors are found, the output looks the same as without the -y option.

So to apply this during reboot, without prompt, you have to Change kernel parameter. **fsck.mode=force** will force a file check.

Steps:

1. open **/etc/default/grub**
2. add the following parameters **fsck.mode=force fsck.repair=yes**

   GRUB_CMDLINE_LINUX_DEFAULT="quiet splash fsck.mode=force fsck.repair=yes"

   the new parameters added here are: **fsck.mode=force fsck.repair=yes**

   caution: Make sure you don't edit anything else, and that the edits you've made are correct, or else your computer may fail to boot

3. **update grub configuration**

   sudo update-grub

In case of network outage

If there has been a network outage in a cluster (e.g. firewall reboot), the following things should be checked to verify they are working:

Check services

```
$ ./manage.py check_services
# or go to
https://[commcarehq.environment.path]/hq/admin/system/check_services
```

Check that change feeds are still processing

You can use this graph on datadog

Service Information

Restarting services: `cchq <env> service <service_name> restart`

Stopping services: `cchq <env> service <service_name> stop`

Service logs: `cchq <env> service <service_name> logs`

Datadog Dashboards

postgres/pgbouncer

redis

rabbitmq

pillow

celery/celerybeat

elasticsearch

kafka

Blob DB Dashboard

couch

formplayer

Switching to Maintenance Page

To switch to the Maintenance page, if you stop all web workers then the proxy will revert to "CommCare HQ is currently undergoing maintenance…".

```
$ cchq <env> service webworker stop
```

To stop all supervisor processes use:

```
$ cchq <env> service commcare stop
```

Couch 2.0

Important note about CouchDB clusters. At Dimagi we run our CouchDB clusters with at least 3 nodes, and **store all data in triplicate**. That means if one node goes down (or even two nodes!), there are no user-facing effects with regards to data completeness so long as no traffic is being routed to the defective node or nodes. However, we have seen situations where internal replication failed to copy some documents to all nodes, causing views to intermittently return incorrect results when those documents were queried.

Thus in most cases, the correct approach is to stop routing traffic to the defective node, to stop it, and then to solve the issue with some better breathing room.

(If you do not store your data in duplicate or triplicate, than this does not apply to your cluster, and a single node being down means the database is either down or serving incomplete data.)

Couch node is down

If a couch node is down, the couch disk might be full. In that case, see *Couch node data disk is full* below. Otherwise, it could mean that the node is slow to respond, erroring frequently, or that the couch process or VM itself in a stopped state.

Monitors are setup to ping the proxy instead of couch instance directly, so the error will appear as "instance:http://**:raw-html-m2r:`<proxy ip>`**/*node/couchdb***:raw-html-m2r:`<couch node ip>`**/".

1. log into couch node ip

2. service couchdb2 status
3. If it's not running start it and begin looking through log files (on the proxy, couch's files, maybe kernel or syslog files as well) to see if you can determine the cause of downtime
4. **If it is running it could just be very slow at responding.**
    a. Use fauxton to see if any tasks are running that could cause couch to become unresponsive (like large indexing tasks)
    b. It could also have ballooned (ICDS) which is out of our control
5. If it's unresponsive and it's out of our control to fix it at the time, go to the proxy machine and comment out the fault node from the nginx config. This will stop sending requests to that server, but it will continue to replicate. When the slowness is over you can uncomment this line and begin proxying reads to it again

Couch node data disk is high

Your best bet if the disk is around 80% is to compact large dbs. If this happens regularly, you're probably better off adding more disk.

Log onto a machine that has access to couchdb:

```
cchq ${env} ssh django_manage
```

and then post to the _compact endpoints of the larger dbs, e.g.:

```
curl -X POST http://${couch_proxy}:25984/commcarehq__auditcare/_compact -v -u ${couch_
↪username} -H 'Content-Type: application/json' -d'{}'
curl -X POST http://${couch_proxy}:25984/commcarehq__receiverwrapper/_compact -v -u $
↪{couch_username} -H 'Content-Type: application/json' -d'{}'
```

where `${couch_proxy}` is the address of the couchdb2_proxy machine (`cchq ${env} lookup couchdb2_proxy`) and ${couch_username} is the value of the `COUCH_USERNAME` secret (`cchq ${env} secrets view COUCH_USERNAME`). You will also need to enter the value of the `COUCH_PASSWORD` secret (`cchq ${env} secrets view COUCH_PASSWORD`).

Couch node data disk is full

If there is more than one couch node, and the other nodes are healthy, the fastest way to get to a calmer place is to pull the node with the full disk out of the proxy so requests stop getting routed to it. First

- Check that the other nodes do not have a full disk

To stop routing data to the node:

1. Comment it out under `[couchdb2]` in the `inventory.ini`
2. Run .. code-block:: bash

    cchq <env> ansible-playbook deploy_couchdb2.yml –tags=proxy

3. Put the node in maintenance mode.
4. Verify internal replication is up to date.
5. Stop its couchdb process .. code-block:: bash

    cchq production run-shell-command <node-name> 'monit stop couchdb2' -b

The steps for this will differ depending on your hosting situation.

Link to internal Dimagi docs on How to modify volume size on AWS.

Once the disk is resized, couchdb should start normally. You may want to immediately investigate how to compact more aggressively to avoid the situation again, or to increase disk on the other nodes as well, since what happens on one is likely to happen on others sooner rather than later barring any other changes.

Once the node has enough disk

1. Start the node (or ensure that it's already started)
2. Force internal replication.
3. Verify internal replication is up to date.
4. Clear node maintenance mode.
5. Reset your inventory.ini to the way it was (i.e. with the node present under the `[couchdb2]` group)
6. Run the same command again to now route a portion of traffic back to the node again: .. code-block:: bash

   cchq <env> ansible-playbook deploy_couchdb2.yml –tags=proxy

Compacting a shard

If a couch node is coming close to running out of space, it may not have enough space to compact the full db. You can start a compaction of one shard of a database using the following:

```
curl "<couch ip>:15986/shards%2F<shard range i.e. 20000000-3fffffff>%2F<database>.<The␣
↪timestamp on the files of the database>/_compact" -X POST -H "Content-Type:␣
↪application/json" --user <couch user name>
```

It's important to use port 15986. This is the couch node endpoint instead of the cluster. The only way to find the timstamp is to go into /opt/data/couchdb2/shards and look for the filename of the database you want to compact

If it's a global database (like _global_changes), then you may need to compact the entire database at once

```
curl "<couch ip>:15984/_global_changes/_compact" -X POST -H "Content-Type: application/
↪json" --user <couch user name>
```

Documents are intermittently missing from views

This can happen if internal cluster replication fails. The precise causes are unknown at time of writing, but it has been observed after maintenance was performed on the cluster where at least one node was down for a long time or possibly when a node was stopped too soon after another node was brought back online after being stopped.

It is recommended to follow the *instructions above* (use maintenance mode and verify internal replication is up to date) when performing cluster maintenance to avoid this situation.

We have developed a few tools to find and repair documents that are missing on some nodes:

```
# Get cluster info, including document counts per shard. Large persistent
# discrepancies in document counts may indicate problems with internal
# replication.
commcare-cloud <env> couchdb-cluster-info --shard-counts [--database=...]

# Count missing forms in a given date range (slow and non-authoritative). Run
```

(continues on next page)

(continued from previous page)

```
# against production cluster. Increase --min-tries value to increase confidence
# of finding all missing ids.
./manage.py corrupt_couch count-missing forms --domain=... --range=2020-11-15..2020-11-
↪18 --min-tries=40

# Exhaustively and efficiently find missing documents for an (optional) range of
# ids by running against stand-alone (non-clustered) couch nodes that have
# snapshot copies of the data from a corrupt cluster. Multiple instances of this
# command can be run simultaneously with different ranges.
./manage.py corrupt_couch_nodes NODE1_IP:PORT,NODE2_IP:PORT,NODE3_IP:PORT forms --
↪range=1fffffff..3fffffff > ~/missing-forms-1fffffff..3fffffff.txt

# Repair missing documents found with previous command
./manage.py corrupt_couch repair forms --min-tries=40 --missing ~/missing-forms-1fffffff.
↪.3fffffff.txt

# See also
commcare-cloud <env> couchdb-cluster-info --help
./manage.py corrupt_couch --help
./manage.py corrupt_couch_nodes --help
```

The process of setting up stand-alone nodes for `corrupt_couch_nodes` will differ depending on the hosting environment and availability of snapshots/ backups. General steps once nodes are setup with snapshots of production data:

- Use a unique Erlang cookie on each node (set in `/opt/couchdb/etc/vm.args`). Do this before starting the couchdb service.
- Remove all nodes from the cluster except local node. The couch_node_standalone_fix.py script can be used to do this.

DefaultChangeFeedPillow is millions of changes behind

Most of our change feed processors (pillows) read from Kafka, but a small number of them serve to copy changes from the CouchDB `_changes` feeds *into* Kafka, the main such pillow being `DefaultChangeFeedPillow`. These pillows store as a checkpoint a CouchDB "seq", a long string that references a place in the _changes feed. While these `seq`s have the illusion of durability (that is, if couch gives you one, then couch will remember it when you pass it back) there are actually a number of situations in which CouchDB no longer recognizes a `seq` that it previously gave you. Two known examples of this are:

- **If you have migrated to a different CouchDB instance using replication, it will *not*** honor a `seq` that the old instance gave you.
- **If you follow the proper steps for draining a node of shards (data) and then remove it,** some `seq`s may be lost.

When couch receives a `seq` it doesn't recognize, it doesn't return an error. Instead it gives you changes *starting at the beginning of time*. This results in what we sometimes call a "rewind", when a couch change feed processor (pillow) suddenly becomes millions of changes behind.

If you encounter a pillow rewind, you can fix it by

- figuring out when the rewind happened,
- finding a recent CouchDB change `seq` from before the rewind happened, and
- resetting the pillow checkpoint to this "good" `seq`

Figure out when the rewind happened

Look at https://app.datadoghq.com/dashboard/ewu-jyr-udt/change-feeds-pillows for the right environment, and look for a huge jump in needs_processed for DefaultChangeFeedPillow.

Find a recent seq

Run

```
curl $couch/commcarehq/_changes?descending=true | grep '"1-"'
```

This will start at the latest change and go backwards, filtering for "1-" because what we want to find is a doc that has only been touched once (so we can easily reason with timestamps in the doc). Start looking up the docs in couch by doc id, until you find a doc with an early enough timestamp (like a form with `received_on`). You're looking for a recent timestamp that happened at a time *before* the rewind happened.

Reset the pillow checkpoint to this "good" seq

Run

```
cchq <env> django-manage shell --tmux
```

to get a live production shell on the `django_manage` machine, and manually change the checkpoint using something like the following lines (using the seq you found above instead, of course):

```
# in django shell
seq = '131585621-g1AAAAKzeJzLYWBg4MhgTmEQTc4vTc5ISXIwNNAzMjDSMzHQMzQ2zQFKMyUyJMn___8_
↪K4M5ieFXGmMuUIw9JdkkxdjEMoVBBFOfqTkuA40MwAYmKQDJJHu4mb_
↪cwWamJZumpiaa49JKyFAHkKHxcEP31oMNNTJMSbIwSCbX0ASQofUwQ3_-
↪uQI21MwkKcnYxAyfoVjCxdIcbGYeC5BkaABSQGPnQxw7yQZibpJpooGFGQ7dxBi7AGLsfrCxfxKPg401MDI2MzClxNgDEGPvQ1zrW
↪'
from pillowtop.utils import get_pillow_by_name
p = get_pillow_by_name('DefaultChangeFeedPillow')
p.checkpoint.update_to(seq)
```

Nginx

Occasionally a staging deploy fails because during a previous deploy, there was an issue uncommenting and re-commenting some lines in the nginx conf.

When this happens, deploy will fail saying

nginx: configuration file /etc/nginx/nginx.conf test failed To fix, log into the proxy and su as root. Open the config and you'll see something like this

```
/etc/nginx/sites-enabled/staging_commcare
#
# Ansible managed, do not edit directly
#

upstream staging_commcare {
```

(continues on next page)

(continued from previous page)

```
  zone staging_commcare 64k;

    least_conn;

#server hqdjango0-staging.internal-va.commcarehq.org:9010;
#server hqdjango1-staging.internal-va.commcarehq.org:9010;
}
```

Ignore the top warning. Uncomment out the servers. Retsart nginx. Run restart_services.

NFS & File serving / downloads

For downloading files we let nginx serve the file instead of Django. To do this Django saves the data to a shared NFS drive which is accessible to the proxy server and then returns a response using the XSendfile/X-Accel-Redirect header which tells nginx to look for the file and serve it to the client directly.

The NFS drive is hosted by the DB machine e.g. hqdb0 and is located at /opt/shared_data (see ansible config for exact name). On all the client machines it is located at /mnt/shared_data (again see ansible config for exact name),

Troubleshooting

It is possible that the mounted NFS folder on the client machines becomes disconnected from the host in which case you'll see errors like "Webpage not available"

To verify that this is the issue log into the proxy machine and check if there are any files in the shared data directories. If there are folders but no files then that is a good indication that the NFS connections has been lost. To re-establish the connection you should unmount and re-mount the drive:

```
$ su
$ umount -l /mnt/shared_data
$ mount /mnt/shared_data
# verify that it is mounted and that there are files in the subfolders
```

It may happen, specially if the client crashes or has kernel oops, that a connection gets in a state where it cannot be broken nor re-established. This is how we force re-connection in a webworker.

1. Verify NFS is actually stuck
    1. `df` doesn't work, it hangs. Same goes for `lsof`.
    2. `umount` results in `umount.nfs: /mnt/shared_icds`: device is busy
2. top all app processes (gunicorn, etc) and datadog
    1. `sudo supervisorctl stop all`
    2. `sudo service datadog-agent stop`
3. Force umount
    1. `sudo umount -f /mnt/shared_icds`
        - if that doesn't work make sure to kill all app processes in e.g. for webworkers `ps aux | grep gunicor[n]`
4. Re-mount
    1. `sudo mount /mnt/shared_icds`

2. Verify NFS mount works: `df`

5. Start supervisor and app processes

    1. `sudo service supervisord start`
    2. `sudo supervisorctl start all`
    3. `sudo service datadog-agent start`

If none of the above works check that nfsd is running on the shared_dir_host.

```
$ ps aux | grep nfsd
$ service nfs-kernel-server status
```

Pgbouncer

We use pgbouncer as a connection pooler for PostgreSQL.

It is configured to use the "transaction" pool mode which means that each server connection is assigned to client only during a transaction. When PgBouncer notices that transaction is over, the server will be put back into pool. This does have some limitations in terms of what the client can do in the connection e.g. no prepared statements. The full list of supported / unsupported operations is found on the pgboucer wiki.

Get a pgbouncer shell

```
$ psql -U {commcarehq-user} -p 6432 pgbouncer
```

Check connection status

```
pgbouncer=# show pools;
  database   |      user      | cl_active | cl_waiting | sv_active | sv_idle | sv_used |␣
↪sv_tested | sv_login | maxwait
-----------+----------------+-----------+------------+-----------+---------+---------+--
↪---------+----------+---------
 commcarehq | ************** |        29 |          0 |        29 |      10 |       8 | ␣
↪       0 |        0 |       0
 pgbouncer  | pgbouncer      |         1 |          0 |         0 |       0 |       0 | ␣
↪       0 |        0 |       0

pgbouncer=# show clients;
 type |      user      |  database  | state  |      addr      | port  |  local_addr   |␣
↪local_port |    connect_time     |    request_time     |    ptr    |   link
------+----------------+------------+--------+----------------+-------+---------------+--
↪----------+---------------------+---------------------+-----------+-----------
 C    | ************** | commcarehq | active | 10.209.128.58  | 39741 | 10.176.193.42 | ␣
↪     6432 | 2015-05-21 12:48:57 | 2015-05-21 13:44:07 | 0x1a59cd0 | 0x1a556c0
 C    | ************** | commcarehq | active | 10.209.128.58  | 40606 | 10.176.193.42 | ␣
↪     6432 | 2015-05-21 13:04:34 | 2015-05-21 13:04:34 | 0x1a668d0 | 0x1a6f590
 C    | ************** | commcarehq | active | 10.177.130.82  | 45471 | 10.176.193.42 | ␣
↪     6432 | 2015-05-21 13:17:04 | 2015-05-21 13:44:21 | 0x1a32038 | 0x1a8b060
 C    | ************** | commcarehq | active | 10.177.130.82  | 45614 | 10.176.193.42 | ␣
```

(continues on next page)

(continued from previous page)

```
↪     6432 | 2015-05-21 13:17:23 | 2015-05-21 13:17:23 | 0x1a645a8 | 0x1a567a0
C    | ************** | commcarehq | active | 10.177.130.82  | 45680 | 10.176.193.42 | ␣
↪     6432 | 2015-05-21 13:17:31 | 2015-05-21 13:44:21 | 0x1a6a110 | 0x1a8a250
```

The columns in the "show pools" output have the following meanings:

cl_active: Connections from clients which are associated with a PostgreSQL connection cl_waiting: Connections from clients that are waiting for a PostgreSQL connection to service them sv_active: Connections to PostgreSQL that are in use by a client connection sv_idle: Connections to PostgreSQL that are idle, ready to service a new client connection sv_used: PostgreSQL connections recently released from a client session. These will end up in sv_idle if they need to once pgbouncer has run a check query against them to ensure they are in a good state. max_wait: The length of time the oldest waiting client has been waiting for a connection

If you want to monitor the connections over a short period of time you can run this command (while logged in as the cchq user): `watch -n 2 pgb_top` You can also access the pgbouncer console easily with this command (while logged in as the cchq user): `pgb`

Postgres Troubleshooting

Common restart problems

If you see something like the following line in the logs:

could not open file ""/etc/ssl/certs/ssl-cert-snakeoil.pem"": Permission denied You can run the following commands to fix it

```
cd /opt/data/postgresql/9.4/main/
chown postgres:postgres server.crt
chown postgres:postgres server.key
```

More information on this error is available here.

Dealing with too many open connections

Sometimes Postgres gets into a state where it has too many open connections. In this state HQ gets very slow and you will see many 500 errors of the form: "OperationalError : FATAL: remaining connection slots are reserved for non-replication superuser connections"

In this case you can check what machines are hogging connections by logging into postgres and using the following steps:

```
$ su
$ sudo -u postgres psql commcarehq
```

```
select client_addr, datname as database, count(*) as total, sum(case when query = '<IDLE>
↪' then 1 else 0 end) as idle from pg_stat_activity group by client_addr, datname;
```

This will print something like the following:

```
  client_addr   | database   | total | idle
----------------+------------+-------+------
                | commcarehq |     4 |    2
```

(continues on next page)

(continued from previous page)

```
 10.209.128.58  | commcarehq |    9 |   5
 10.177.130.82  | commcarehq |    7 |   7
 10.208.22.37   | commcarehq |    6 |   5
 10.223.145.60  | commcarehq |    1 |   0
 10.208.148.179 | commcarehq |    3 |   3
 10.176.132.63  | commcarehq |   24 |   23
 10.210.67.214  | commcarehq |    3 |   2
```

When using pgbouncer the following command can be used to list clients:

```
$ psql -h localhost -p 6432 -U $USERNAME pgbouncer -c "show clients" | cut -d'|' -f 5 |␣
↪tail -n +4 | sort | uniq -c
    10  10.104.103.101
     5  10.104.103.102
     2  10.104.103.104
```

To see a list of queries (ordered by the long running ones first) you can do something like the following. This can also be exported to csv for further analysis.

```
  SELECT pid, datname, query_start, now() - query_start as duration, state, query as␣
↪current_or_last_query FROM pg_stat_activity WHERE state = 'active' OR query_start >␣
↪now() - interval '1 min' ORDER BY state, query_start;

This can also be exported to csv for further analysis.
```

```
Copy (SELECT state, query_start, client_addr, query FROM pg_stat_activity ORDER BY query_
↪start) TO '/tmp/pg_queries.csv' WITH CSV;
```

Use `iotop` to see what processes are dominating the IO and get their process IDs.

```
SELECT pid, query_start, now() - query_start as duration, client_addr, query FROM pg_
↪stat_activity WHERE procpid = {pid} ORDER BY query_start;
```

DO NOT EVER ``kill -9`` any PostgreSQL processes. It can bring the DB process down.

This shouldn't be necessary now that we've switched to using pgbouncer (but it still is currently!).

After checking open connections you can kill connections by IP address or status. The following command will kill all open IDLE connections from localhost (where pgbouncer connections route from) and is a good way to reduce the load:

Kill all idle connections

```
SELECT pg_terminate_backend(procpid) FROM pg_stat_activity WHERE client_addr = '127.0.0.1
↪' AND query = '<IDLE>';
```

Kill a single query

```
SELECT pg_terminate_backend({procpid})
```

Replication Delay

https://www.enterprisedb.com/blog/monitoring-approach-streaming-replication-hot-standby-postgresql-93

- Check if wal receiver and sender process are running respectively on standby and master using `ps aux | grep receiver` and `ps aux | grep sender`
- Alternatively use SQL `select * from pg_stat_replication` on either master or standby
- If WAL processes are not running, check logs, address any issues and may need to reload/restart postgres
- Check logs for anything suspicious
- Checking replication delay
  - Use datadog
  - Run queries on nodes:

```
--- on master
select
  slot_name,
  client_addr,
  state,
  pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(), sent_location))␣
↪sending_lag,
  pg_size_pretty(pg_xlog_location_diff(sent_location, flush_location)) receiving_lag,
  pg_size_pretty(pg_xlog_location_diff(flush_location, replay_location)) replaying_lag,
  pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(), replay_location))␣
↪total_lag
from pg_replication_slots s
left join pg_stat_replication r on s.active_pid = r.pid
where s.restart_lsn is not null;

-- On standby

SELECT now() - pg_last_xact_replay_timestamp() AS replication_delay;
```

In some cases it may be necessary to restart the standby node.

PostgreSQL disk usage

Use the following query to find disc usage by table where child tables are added to the usage of the parent.

```
SELECT *, pg_size_pretty(total_bytes) AS total
    , pg_size_pretty(index_bytes) AS INDEX
    , pg_size_pretty(toast_bytes) AS toast
    , pg_size_pretty(table_bytes) AS TABLE
  FROM (
  SELECT *, total_bytes-index_bytes-COALESCE(toast_bytes,0) AS table_bytes FROM (
```

(continues on next page)

(continued from previous page)

```
      SELECT c.oid,nspname AS table_schema, relname AS TABLE_NAME
              , c.reltuples AS row_estimate
              , pg_total_relation_size(c.oid) AS total_bytes
              , pg_indexes_size(c.oid) AS index_bytes
              , pg_total_relation_size(reltoastrelid) AS toast_bytes
          FROM pg_class c
          LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
          WHERE relkind = 'r'
  ) a
) a order by total_bytes desc;
```

```
SELECT
  main_table,
  row_estimate,
  pg_size_pretty(total_size) as total,
  pg_size_pretty(index_bytes) as index,
  pg_size_pretty(toast_bytes) as toast
FROM (
SELECT
    CASE WHEN HC.inhrelid IS NOT NULL THEN CP.relname
        ELSE C.relname END as main_table,
    sum(C.reltuples) AS row_estimate,
    sum(pg_total_relation_size(C.oid)) AS "total_size",
    sum(pg_indexes_size(C.oid)) AS index_bytes,
    sum(pg_total_relation_size(C.reltoastrelid)) AS toast_bytes
FROM pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
LEFT JOIN pg_inherits HC ON (HC.inhrelid = C.oid)
LEFT JOIN pg_class CP ON (HC.inhparent = CP.oid)
WHERE nspname NOT IN ('pg_catalog', 'information_schema')
    AND C.relkind <> 'i' AND C.relkind <> 'S' AND C.relkind <> 'v'
    AND nspname !~ '^pg_toast'
GROUP BY main_table
ORDER BY total_size DESC
) as a;
```

```
SELECT
    CASE WHEN HC.inhrelid IS NOT NULL THEN CP.relname
        ELSE C.relname END as main_table,
    sum(seq_scan) as seq_scan,
    sum(seq_tup_read) as seq_tup_read,
    sum(idx_scan) as idx_scan,
    sum(idx_tup_fetch) as idx_tup_fetch,
    sum(n_tup_ins) as n_tup_ins,
    sum(n_tup_upd) as n_tup_upd,
    sum(n_tup_del) as n_tup_del,
    sum(n_tup_hot_upd) as n_tup_hot_upd,
    sum(n_live_tup) as n_live_tup,
    sum(n_dead_tup) as n_dead_tup
FROM pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
```

(continues on next page)

(continued from previous page)

```
LEFT JOIN pg_inherits HC ON (HC.inhrelid = C.oid)
LEFT JOIN pg_class CP ON (HC.inhparent = CP.oid)
LEFT JOIN pg_stat_user_tables t ON (C.oid = t.relid)
WHERE nspname NOT IN ('pg_catalog', 'information_schema')
    AND C.relkind <> 'i' AND C.relkind <> 'S' AND C.relkind <> 'v'
    AND nspname !~ '^pg_toast'
GROUP BY main_table
ORDER BY n_tup_ins DESC;
```

In Case PostgreSQL fails with `No space left on device` error message and in order to free up space needed to restart the PostgreSQL then take the following steps

- Stop the `Pgbouncer`
- There is a dummy file of 1GB placed in encrypted root path `/opt/data/emerg_delete.dummy` which can be deleted.
- It will give you enough space to restart Database. Reclaim the disk space.
- Start the `Pgbouncer`
- Once the issue has been resolved you should re-add the dummy file for future use: .. code-block:

```
dd if=/dev/zero of=/opt/data/emerg_delete.dummy count=1024 bs=1048576
```

At all the times, PostgreSQL maintains a write-ahead log (WAL) in the pg_xlog/ for version <10 and in pg_wal/ for version >=10 subdirectory of the cluster's data directory. The log records for every change made to the database's data files. These log messages exists primarily for crash-safety purposes.

It contains the main binary transaction log data or binary log files. If you are planning for replication or Point in time Recovery, we can use this transaction log files.

We cannot delete this file. Otherwise, it causes a database corruption. The size of this folder would be greater than actual data so If you are dealing with massive database, 99% chance to face disk space related issues especially for the pg_xlog or pg_wal folder.

There could be multiple reason for folder getting filled up.

- Archive Command is failing.
- Replication delay is high.
- Configuration params on how much WAL logs to keep.

If you are able to fix the above related , then logs from this folder will be cleared on next checkpoints.

If it's absolutely necessary to delete the logs from this folder. Use following commands to do it. (Do not delete logs from this folder manually)

```
# you can run this to get the latest WAL log
/usr/lib/postgresql/<postgres-version>/bin/pg_controldata /opt/data/postgresql/<postgres-
↪version>/main

Deleting
/usr/lib/postgresql/<postgres-version>/bin/pg_archivecleanup -d /opt/data/postgresql/
↪<postgres-version>/main/<pg_wal|| pg_xlog> <latest WAL log filename>
```

Celery

Check out *Celery Firefighting Guide*.

Monitoring

Sometimes it's helpful to check "Flower", a UI for celery monitoring http://hqcelery1.internal-va.commcarehq.org:5555/ (you have to be VPN'd in).

You can also check the current celery queue backlogs on datadog. Any spikes indicate a backup, which may result in things like delays in sending scheduled reports or exports. If there's a problem, there should also be an alert here and on #hq-ops on Slack.

Also, see the bottom of this page for some useful firefighting commands.

Celery consuming all the disk space

On occasion, the celery_taskmeta table grows out of control and takes up all the disk space on the database machine very quickly. Often one our disk space monitors will trip when this happens. To diagnose and ensure that the it is indeed the celery_taskmeta table that has grown too large, you can run the above Postgres command to check disk usage by table.

To fix the issue, you can then run these commands in a psql shell after stopping the Celery workers:

```
# Ensure Celery workers have been stopped
truncate celery_taskmeta;
vacuum full celery_taskmeta;
# Start Celery workers again
```

Elasticsearch

Check Cluster Health

It's possible to just ping (a) server(s):

```
$ curl -XGET 'http://es[0-3].internal-icds.commcarehq.org:9200/'
{
  "status" : 200,
  "name" : "es0",
  "cluster_name" : "prodhqes-1.x",
  "version" : {
    "number" : "1.7.3",
    "build_hash" : "05d4530971ef0ea46d0f4fa6ee64dbc8df659682",
    "build_timestamp" : "2015-10-15T09:14:17Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.4"
  },
  "tagline" : "You Know, for Search"
}
```

Or check for health:

```
$ curl -XGET 'http://es0.internal-icds.commcarehq.org:9200/_cluster/health?pretty=true'
{
  "cluster_name" : "prodhqes-1.x",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 4,
  "number_of_data_nodes" : 4,
  "active_primary_shards" : 515,
  "active_shards" : 515,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0
}
```

Data missing on ES but exist in the primary DB (CouchDB / PostgreSQL)

We've had issues in the past where domains have had some of their data missing from ES. This might correlate with a recent change to ES indices, ES-related upgrade work, or ES performance issues. All of these instabilities can cause strange flaky behavior in indexing data, especially in large projects.

First, you need to identify that this issue is not ongoing and widespread.

1) visit the affected domain's Submit History or Case List report to verify that recent submissions are still being indexed on ES (if they are in the report, they are in ES) 2) check the modification date of the affected data and then check the reports around that date and surrounding dates. 3) spot check another domain with a lot of consistent submissions to see if there are any recent and past issues surrounding the reported affected date(s).

If you don't see any obvious issues, it's likely a strange data-flakiness issue. This can be resolved by running the following management commands (run in a tmux since they may take a long time to complete):

```
pthon manage.py stale_data_in_es [form/case] --domain <domain> [--start=YYYY-MM-DD] [--
↪end=YYYY-MM-DD] > stale_data.tsv
pthon manage.py republish_doc_changes stale_data.tsv
```

You can also do a quick analysis of the output data to find potentially problematic dates:

```
cat state_data.tsv | cut -f 6 | tail -n +2 | cut -d' ' -f 1 | uniq -c

      2 2020-10-26
    172 2020-11-03
     14 2020-11-04
```

If you DO see obvious issues, it's time to start digging for problematic PRs or checking performance monitoring graphs.

Low disk space free

"[INFO][cluster.routing.allocation.decider] [hqes0] low disk watermark [85%] exceeded on … replicas will not be assigned to this node"

is in the logs, then ES is running out of disk space. If there are old, unused indices, you can delete them to free up disk space.

```
$ ./manage.py prune_elastic_indices --delete
Here are the indices that will be deleted:
hqapps_2016-07-08_1445
hqusers_2016-02-16_1402
report_xforms_20160707_2322
xforms_2016-06-09
```

Hopefully there are stale indices to delete, otherwise you'll need to investigate other options, like increasing disk size or adding a node to the cluster. Be sure to check the disk usage after the deletions have completed.

Request timeouts

"ESError: ConnectionTimeout caused by - ReadTimeoutError(HTTPConnectionPool(host='hqes0.internal-va.commcarehq.org', port=9200): Read timed out. (read timeout=10))"

This could be caused by a number of things but a good process to follow is to check the ES dashboard on Datadog and the slow logs on the ES machines:

```
# folder + filename may not be exact
$ tail -f /opt/data/elasticsearch-1.7.1/logs/prodhqes-1.x_index_search_slowlog.log
```

Unassigned shards

Currently on ICDS (maybe on prod/india) shard allocation is disabled. In case a node goes down all the shards that were on the node get unassigned. Do not turn on automatic shard allocation immediately since that might cause lot of unexpected shards to move around. Instead follow below instructions (the last point is very important for large ES clusters)

- Check which nodes are down and restart them.
- Once all nodes are up, all the primary nodes should automatically get assigned.
  - Shard assignment can be checked via Elasticsearch shards API or the shards graph on Elasticsearch datadog dashboard
- If any primaries are not allocated. Use rereoute API (official docs)
  - Reroute according to existing shard allocation
  - The rerouting of unassigned primary shards will cause data loss (w.r.t es_2.4.6). **:raw-html-m2r:`<br>`** :warning: The **:raw-html-m2r:`<b>allow_primary</b>`** parameter will force a new empty primary shard to be allocated without any data. If a node which has a copy of the original shard (including data) rejoins the cluster later on, that data will be deleted: the old shard copy will be replaced by the new live shard copy.
  - Example reroute command to allocate replica shard

```
curl -XPOST 'http://<es_url>/_cluster/reroute' -d ' {"commands" :[{"allocate": {
↪"shard": 0, "node": "es34", "index": "xforms_2020-02-20"}}]}'
```

- Replicas won't get auto assigned. To assign replicas, auto shard allocation needs to be enabled
  - Make sure no primaries are unassigned
  - Turn on auto shard allocation using .. code-block:

    ```
    curl 'http://<es_url>/_cluster/settings/' -X PUT  --data '{"transient":{
    ↪"cluster.routing.allocation.enable":"all"}}'
    ```

  - Wait for replicas to get assigned.
- Finally **remember to turn off** auto shard allocation using .. code-block:

  ```
  curl 'http://<es_url>/_cluster/settings/' -X PUT  --data '{"transient":{"cluster.
  ↪routing.allocation.enable":"none"}}'
  ```

```
curl -XPUT '<es url/ip>:9200/_cluster/settings' -d '{ "transient":
  { "cluster.routing.allocation.enable" : "all"
  }
}'
# wait for shards to be allocated
./scripts/elasticsearch-administer.py <es url> shard_status # all shards should say␣
↪STARTED
curl -XPUT '<es url/ip>:9200/_cluster/settings' -d '{ "transient":
  { "cluster.routing.allocation.enable" : "none"
  }
}'
./manage.py ptop_reindexer_v2 <index that had the unassigned shards> # run this in a␣
↪tmux/screen session as it will likely take a while
./scripts/elasticsearch-administer.py <es url> shard_status # the shards for indexes␣
↪that had unassigned shards should have around the same number of docs
```

Make sure to run the management command in the most recent release directory (may not be current if this failed before the entire deploy went through)

Redis

Understanding the Top 5 Redis Performance Metrics

Selectively flushing keys

Sometimes in order for a fix to propagate you'll need to flush the cache for certain keys. You can use this script to selectively flush keys by globbing.

```
redis-cli
127.0.0.1:6379> EVAL "local keys = redis.call('keys', ARGV[1]) \n for i=1,#keys,5000 do \
↪n redis.call('del', unpack(keys, i, math.min(i+4999, #keys))) \n end \n return keys" 0␣
↪unique-cache-key*
```

A lot of times Redis will prefix the cache key with something like `:1:` so you'll often need to do *unique-cache-key*

Disk full / filling rapidly

We have seen a situation where the redis disk fills up with files of the pattern /opt/data/redis/temp-rewriteaof-*.aof. This happens when redis maxmemory is configured to be too high a proportion of the total memory (although the connection is unclear to the author, Danny). This blog http://oldblog.antirez.com/post/redis-persistence-demystified.html/ explains AOF rewrite files. The solution is to (1) lower maxmemory and (2) delete the temp files.

```
root@redis0:/opt/data/redis# cd /opt/data/redis/
root@redis0:/opt/data/redis# redis-cli
127.0.0.1:6379> CONFIG SET maxmemory 4gb
OK
(1.06s)
root@redis0:/opt/data/redis# rm temp-rewriteaof-\*.aof
```

We use the default AOF auto-rewrite configuration, which is to rewrite the AOF (on-disk replica of in-memory redis data) whenever it doubles in size. Thus disk usage will sawtooth between X and 3X where X is the size of the AOL after rewrite: X right rewrite, 2X when rewrite is triggered, and 3X when the 2X-sized file has been written to a 1X-sized file, but the 2X-sized file has not yet been deleted, followed finally again by X after rewrite is finalized and the old file is deleted.

Since the post-rewrite AOF will only ever contain as much data as is contained in redis memory, and the amount of data contained in redis memory has an upper bound of the maxmemory setting, you should make sure that your disk is at least 3 * maxmemory + whatever the size of the OS install is. Since disk is fairly cheap, give it a comfortable overhead for log files etc.

Checking redis after restart

Redis takes some time to read the AOF back into memory upon restart/startup. To check if it's up, you can run the following:

```
$ cchq <env> ssh ansible@redis

% redis-cli
127.0.0.1:6379> ping
(error) LOADING Redis is loading the dataset in memory
127.0.0.1:6379> ping
(error) LOADING Redis is loading the dataset in memory
127.0.0.1:6379> ping
PONG
```

once it responds with `PONG` redis is back up and ready to serve requests.

Tail the redis log

To show the last few lines of the redis log during firefighting you can run:

```
cchq <env> run-shell-command redis 'tail /opt/data/redis/redis.log'
```

Pillows / Pillowtop / Change feed

Symptoms of pillows being down:

- Data not appearing in reports, exports, or elasticsearch
- UCR or report builder reports behind
- Datadog monitor

Resources:

- graph of change feed activity
- Pillows documentation
- Pillows overview and introduction

Managing Pillows

You can check on the status of the pillow processors with

```
cchq <env> service pillowtop status
```

and you can restart a pillow which is not currently `RUNNING` with

```
cchq <env> service pillowtop start --only=<pillow_name>
```

Note that the elements returned by the `status` command are the names of the processors, not the names of the pillows themselves.

For example if the status command identified that `myenv-production-DefaultChangeFeedPillow-0` was not running, to restart the pillow one would run

```
#Correct - Restarting by pillow name
cchq myenv service pillowtop start --only=DefaultChangeFeedPillow
```

rather than

```
#Incorrect - Restarting by processor name
cchq myenv service pillowtop start --only=myenv-production-DefaultChangeFeedPillow-0
```

Formplayer / Cloudcare / Webapps

Formplayer sometimes fails on deploy due to a startup task (which will hopefully be resolved soon). The process may not fail, but formplayer will still return failure responses. You can try just restarting the process with `sudo supervisorctl restart all` (or specify the name if it's a monolithic environment)

A formplayer machine(s) may need to be restarted for a number of reasons in which case you can run (separate names by comma to run on multiple machines):

```
cchq <env> service formplayer restart --limit=formplayer_bXXX
```

Lock issues

If there are many persistent lock timeouts that aren't going away by themselves, it can be a sign of a socket connection hanging and Java not having a timeout for the connection and just hanging.

In that case, it can be helpful to kill the offending socket connections. The following command queries for socket connections that look like the ones that would be hanging and kills them:

```
cchq <env> run-shell-command formplayer 'ss src {{ inventory_hostname }} | grep ESTAB |␣
↪grep tcp | grep ffff | grep https | cut -d: -f5 | cut -d -f1 | xargs -n1 ss -K sport =
↪' -b
```

Because it's filtered, it won't kill *all* socket connections, but it will kill more socket connections than strictly necessary, since it is difficult to determine which specific connections are the problematic ones. But anecdotally this doesn't cause any user-facing problems. (I still wouldn't do it unless you have to to solve this issue though!)

Full Drives / Out of Disk Space

If disk usage on the proxy ever hits 100%, many basic functions on the machine will stop working. Here are some ways to buy some time.

Basic Commands

You can probe statistics before taking any action. *df -h* or *dh -h /* will show total disk usage. To query specific directory/file usage, use: *du -hs <path>*. Note that these commands still respect permissions. If you need elevated permissions, you can ssh to the affected machine as the ansible user (cchq –control <env> ssh ansible@<machine>), and from there you can use sudo. The ansible password can be found within 1Pass

Clean Releases

Each release / copy of our commcare-hq git repo can be 500M - 2.7G (variation can depend on how efficiently git is storing the history, etc.). It's always safe to run

`$ cchq <env> clean-releases` and sometimes that alone can clean up space. This is run on every deploy, so if you just deployed successfully, don't bother.

Move logs to another drive

Check the size of the log files stored at /home/cchq/www/**:raw-html-m2r:`<environment>`**/log, these can get out of hand. Last time this ocurred, we moved these into the shared drive, which had plenty of available disk space (but check first!)

```
$ mv -v pattern-matching-old-logs.*.gz /mnt/shared/temp-log-storage-main-hd-full/
```

Clear the apt cache

Apt stores its files in */var/cache/apt/archives*. Use *du* as describe above to determine if this cache is consuming too much space. If it is, these files can be cleaned via *apt-get clean* `

`$ apt-get autoremove` This removes packages that are no longer required. Sometimes the savings can be substantial. If you run the above command, it should show you how much space it expects to free up, before you commit to running it. On a recent occasion, this freed up about 20% of the disk

Manually rotate syslog

If for some reason syslog is either not rotating logs or the latest log has grown more than expected you can run

```
mv syslog other.syslog
kill -HUP <pid of syslog>
gzip other.syslog
```

Look at temp folders

On celery machines, specifically, tasks can generate a large volume of temporary files. Use *du* against */opt/tmp* and compare these results to other machines to determine if this is the likely issue. If so, some of these files may still be in use. These files will likely be cleared once the task has completed. If not, we have an automated task that cleans up files older than 2 days. If disk space is in a dire situation, it may be possible to remove some of the older files (using *lsof <file>* or *lsof +D <directory>* can help find what files are in use)

Network Connection Issues (please help expand)

If you suspect some sort of network issue is preventing two servers from talking to each other, the first thing you should verify is that the processes you expect to talk to each other are actually running. After that, here are some things to try:

Ping

Try pinging the machine from the computer you'd expect to initiate the connection. If that doesn't work, try pinging from your local machine while on the VPN.

```
esoergel@hqproxy0:~$ ping hqdb0.internal-va.commcarehq.org
PING hqdb0.internal-va.commcarehq.org (172.24.32.11) 56(84) bytes of data.
64 bytes from 172.24.32.11 (172.24.32.11): icmp_seq=1 ttl=64 time=42.6 ms
64 bytes from 172.24.32.11 (172.24.32.11): icmp_seq=2 ttl=64 time=41.3 ms
```

netcat

Netcat is a mini server. You can set it up to listen on any port and respond to requests. Run something like this to listen on port 1234 and wait for a request:

```
esoergel@hqdb0:~$ printf "Can you see this?" | netcat -l 1234
```

Then go over to the other machine and try to hit that server:

```
$ curl hqdb0.internal-va.commcarehq.org:1234
Can you see this?$
```

Looks like the request went through! If you go back and check on the netcat process, you should see the request:

```
esoergel@hqdb0:~$ printf "Can you see this?" | netcat -l 1234
HEAD / HTTP/1.1
Host: hqdb0.internal-va.commcarehq.org:1234
User-Agent: curl/7.50.1
Accept: */*

esoergel@hqdb0:~$
```

Tips and Tricks

Never run that painful sequence of sudo -u cchq bash, entering the venv, cd'ing to the directory, etc., again just to run a management command. Instead, just run e.g.:

```
cchq <env> django-manage shell --tmux
```

first thing after logging in.

Some Short Write-ups and Examples

See Troubleshooting Pasteboard / HQ chat dumping ground. There is also some ElasticSearch material

Backups

Information for restoring elasticsearch and postgres from a backup are at Restoring From Backups

SMS Gateways

See the page on SMS Gateway Technical Info for API docs and support contacts for each gateway.

9.3.2 Celery Firefighting Guide

Queue is blocked

Symptoms

You check /serverup.txt?only=celery and see a queue has been blocked for some duration. Example of what this looks like:

- Failed Checks (web8-production): celery: reminder_case_update_queue has been blocked for 0:27:57.285204 (max allowed is 0:15:00)

Resolution

You can restart the blocked queue using:

```
cchq <env> service celery restart --only=<celery_queue>
```

To obtain a list of queues run:

```
cchq <env> service celery help
```

Worker is down

Symptoms

You check /hq/admin/system/check_services or /serverup.txt?heartbeat (example: https://www.commcarehq.org/serverup.txt?heartbeat) and it shows the worker is down:

- celery@hqcelery2.internal-va.commcarehq.org_main.1491657794_timestamp worker is down Using the environments and inventory files to find which machine hosts the worker, you log in and verify the worker is stopped:

```
dogeillionaire@hqcelery2:~$ sudo supervisorctl status
commcare-hq-production-celery_background_queue RUNNING    pid 10464, uptime 0:45:47
commcare-hq-production-celery_main STOPPED Apr 08 02:18 PM
commcare-hq-production-celery_saved_exports_queue RUNNING    pid 10463, uptime 0:45:47
```

Resolution

Start the worker:

```
dogeillionaire@hqcelery2:~$ sudo supervisorctl start commcare-hq-production-celery_main
commcare-hq-production-celery_main: started
```

Verify the worker is running:

```
dogeillionaire@hqcelery2:~$ sudo supervisorctl status
commcare-hq-production-celery_background_queue RUNNING    pid 10464, uptime 0:45:47
commcare-hq-production-celery_main RUNNING    pid 10462, uptime 0:01:22
commcare-hq-production-celery_saved_exports_queue RUNNING    pid 10463, uptime 0:45:47
```

Worker won't start

Symptoms

You check /hq/admin/system/check_services or /serverup.txt?heartbeat (example: https://www.commcarehq.org/serverup.txt?heartbeat) and it shows the worker is down:

- celery@hqcelery2.internal-va.commcarehq.org_main.1491657794_timestamp worker is down Using the environments and inventory files to find which machine hosts the worker, you log in and verify the worker has not been able to start:

```
dogeillionaire@hqcelery2:~$ sudo supervisorctl status
commcare-hq-production-celery_background_queue RUNNING    pid 10464, uptime 0:45:47
commcare-hq-production-celery_main FATAL
commcare-hq-production-celery_saved_exports_queue RUNNING    pid 10463, uptime 0:45:47
```

Resolution

View the log file to see what the error is preventing the worker from starting and resolve that error. The log file name and location are given in the service template for supervisor.

`dogeillionaire@hqcelery2:/home/cchq/www/production/log$ cat celery_main.log | less`

> When viewing output with less, pressing shift+G takes you to the end of the file, just pressing G takes you to the beginning of the file, and page up / down scrolls pages.

The error might be an exception in python code raised when starting the worker up, in which case a code fix is needed. Sometimes the error might also be an out of memory issue. If it is an out of memory issue, you can check to see if the machine is still out of memory with htop. If the machine is still out of memory right now, you may need to look for an zombie celery processes from stale child worker processes that were not stopped at previous shutdown using `ps -ef | grep celery` and stop them with `kill` as the cchq user.

Once the error is fixed, follow the instructions under "Worker is down" to start the worker.

Worker did not shut down properly

Symptoms

You check /serverup.txt?heartbeat (example: https://www.commcarehq.org/serverup.txt?heartbeat) and it shows the worker is running when it shouldn't be:

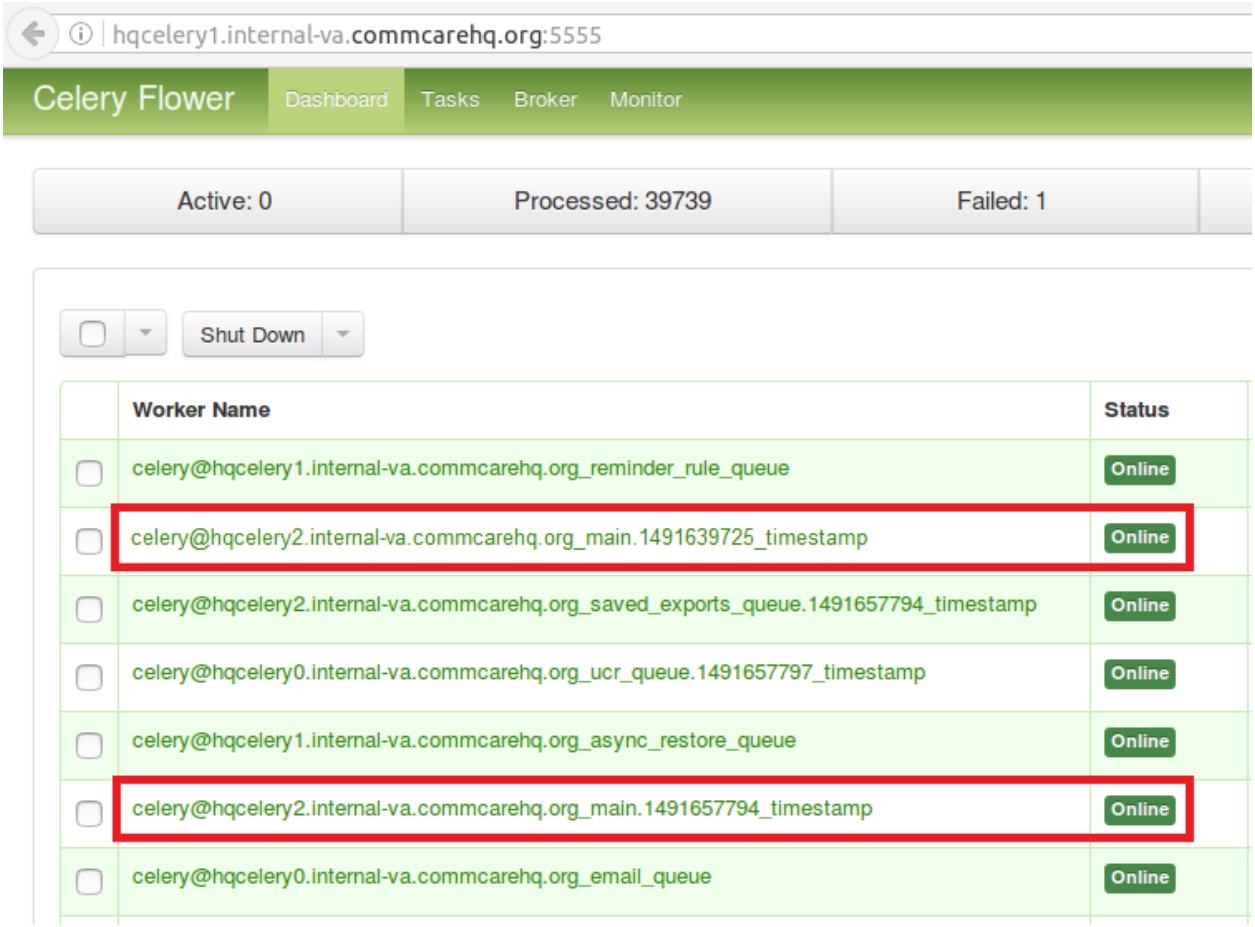
- celery@hqcelery2.internal-va.commcarehq.org_main.1491639725_timestamp celery worker is running when we expect it to be stopped

Resolution

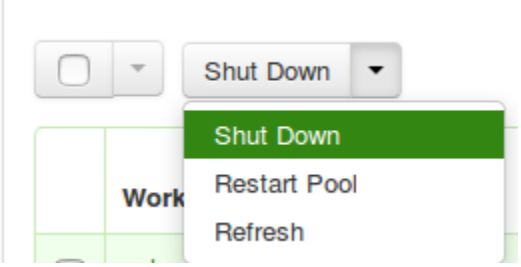
To kill the workers that didn't shut down properly, you can use the `commcare-cloud <env> kill-stale-celery-workers`. This will automatically figure out which ones to kill. If that still doesn't work, follow the steps below.

Using the environments and inventory files to find which machine hosts flower, use your browser to hit port 5555 on that machine (example: http://hqcelery1.internal-va.commcarehq.org:5555/) to view flower.

On the dashboard you should see two of the same workers listed as being online, but with different timestamps in their name:



Check the box next to the worker you saw in the serverup notice (which should also be the one with the older, or smaller, timestamp), and shut it down by selecting Shut Down from the dropdown at the top of the page:



Worker is deadlocked

Symptoms

The worker is running (so there is no down notice), but it won't accept new tasks. If the main worker is deadlocked, people may be reporting that they can't do exports or imports of data. When you view the current active tasks for the worker with the show_celery_tasks management command, it either shows no tasks or tasks that are hours old.

Resolution

Restart the worker:

```
dogeillionaire@hqcelery2:~$ sudo supervisorctl restart commcare-hq-production-celery_main
commcare-hq-production-celery_main: stopped
commcare-hq-production-celery_main: started
```

Verify the worker is running:

```
dogeillionaire@hqcelery2:~$ sudo supervisorctl status
commcare-hq-production-celery_background_queue RUNNING    pid 10464, uptime 0:45:47
commcare-hq-production-celery_main RUNNING    pid 10462, uptime 0:01:22
commcare-hq-production-celery_saved_exports_queue RUNNING    pid 10463, uptime 0:45:47
```

The queue the worker is consuming from has a large backlog of tasks

Symptoms

The datadog monitor for queued tasks has given an alert for the queue that the worker consumes from.

If the main queue has a large backlog of tasks, people may be reporting that they can't do exports or imports of data.

When you view the current active tasks for the worker with the show_celery_tasks management command, it shows tasks that are relatively fresh, so you don't believe the worker is deadlocked.

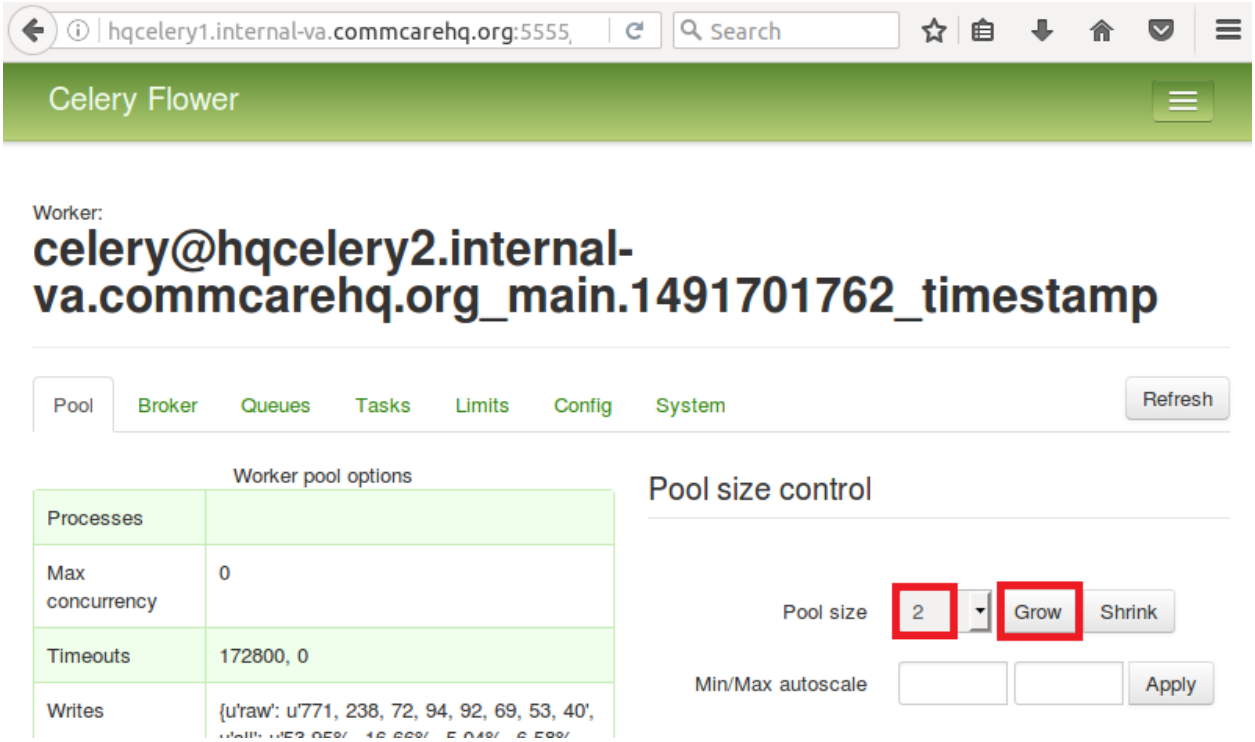
Resolution

For the most part, we just have to wait until the tasks are processed. If it's impacting something like exports/imports, It's worth trying to estimate how long it will take and put up a banner mentioning exports/imports are down at the moment and to not keep retrying them as it will just exacerbate the issue.

If this happens often for the same queue, then it means a longer-term solution is needed, such as increasing the concurrency on the worker, reducing the time it takes for the tasks to be completed, or moving the tasks to a different queue (or to a new queue and worker). However, there are a couple short-term things we may be able to do to help reduce the time we need to wait before it's back to normal:

1. If you log into the machine where the worker is hosted and there is a good amount of free memory (at least 2GB or so), you can temporarily increase the concurrency on the worker. To do this:
    a. Using the environments and inventory files to find which machine hosts flower, use your browser to hit port 5555 on that machine (example: http://hqcelery1.internal-va.commcarehq.org:5555/) to view flower.
    b. From the dashboard, click the name of the worker which consumes from the queue that is backed up.

c. Under "Pool size control", increase the number of child processes that worker has by selecting a number of processes to increase by in the dropdown and click the "Grow" button. For example, if the current concurrency is 4 and you select 2 in the dropdown and click "Grow", the new max concurrency will be 6.



Be careful with this - if you increase by too much you may start to see tasks failing with out of memory (SIGSEGV) errors. Rule of thumb is to only increase by 2 processes per 1 GB of memory you can use up, and always try to leave at least 1 GB of memory free on the machine at all times. So if there's 2 GB of memory free, only increase by 2, and if there's 3 GB of memory free, only increase by 4. If you start having out of memory issues after you do this, you'll need to either shrink the pool or restart the worker.

This rule of thumb also does not apply to the workers that use gevent pooling - we can be a little more liberal about increasing the concurrency on those, keeping in mind that whatever you increase it to, that many threads may be running at a time.

Also note this only temporary; once the worker is restarted on next deploy or manually, it will go back to its old concurrency setting.

1. **If there are a lot of tasks clogging up the queue that are not worth processing anymore (for example, exports that people had initiated long ago that they are no longer waiting for), you can revoke those tasks. To do this, do the following:**

    a. Log into any machine on the cluster (it doesn't have to be where the worker is hosted), and prep the environment as you would for entering a django shell or running any management command:

```
dogeillionaire@hqcelery0:~$ sudo -u cchq bash
cchq@hqcelery0:~$ cd /home/cchq/www/production/current
cchq@hqcelery0:/home/cchq/www/production/current$ source python_env/bin/activate
(python_env) cchq@hqcelery0:/home/cchq/www/production/current$
```

```
b. Invoke the revoke_celery_tasks management command, passing the fully qualified task␣
↪names to revoke as args:
```

```
(python_env) cchq@hqcelery0:/home/cchq/www/production/current$ python manage.py revoke_
↪celery_tasks corehq.apps.export.tasks.populate_export_download_task
2017-04-09 12:34:19.525830 Revoked 161a7623a3f444e7b361da4b4fa6fc42 corehq.apps.export.
↪tasks.populate_export_download_task
2017-04-09 12:34:26.803201 Revoked a855bac716ca4850899866cc97076c3d corehq.apps.export.
↪tasks.populate_export_download_task
```

This command will just keep running, revoking all existing and new tasks that it finds that match the given task name(s). This command is only able to revoke tasks received by the worker from rabbitmq. The worker does not see all the tasks in the queue all at once since the tasks are prefetched by the worker from rabbitmq a little at a time, so to revoke them all you just have to keep it running. When you no longer need it, just stop it with Ctrl+C.

Intermittent datadog connection errors

Symptoms

Receiving alerts that the datadog agent on a celery machine is not reporting. The alerts recover on their own but continue to trigger.

Resolution

This is only relevant if these alerts are for the first celery machine `celery[0]`:

```
cchq <env> service celery restart --limit=celery[0]
```

Common RabbitMQ Firefighting Scenarios

RabbitMQ is down

Symptoms

There are 500 emails saying Connection Refused to a service running on port 5672

You see errors mentioning a celery worker cannot connect to amqp broker in the celery logs

Resolution

See Restarting Services on this *Firefighting Guide*.

Disk filling up

Symptoms

Disk usage warning

Resolution

1. Use 'ncdu' on the machine to detemine if it's RabbitMQ that's using up the disk
2. Check the RabbitMQ dashboard to determine which queue is causing the issue a. https://app.datadoghq.com/screen/integration/237/rabbitmq—overview
3. Ensure that the celery workers are running and consuming the queue
4. Purge the queue. *Only do this if the tasks can be re-queued e.g. pillow_retry_queue*

```
celery -A corehq purge -Q queue_1,queue_2
```

Useful Celery Commands

Show celery tasks

Unfortunately, flower often times will show stale data. To view the most current information on active, reserved, or scheduled tasks for a worker, use this command.

```
python manage.py show_celery_tasks <worker name> <task state>
```

This command prints the celery tasks in the given state on the given worker. For example, to show all active tasks being processed by the main worker:

```
python manage.py show_celery_tasks celery@hqcelery2.internal-va.commcarehq.org_main.
1491701762_timestamp active
```

To view a list of worker names, use the show_celery_workers command.

Show celery workers

To get a quick list of workers that are currently running, use this command:

```
python manage.py show_celery_workers
```

Shut down a celery worker

To initiate a warm shutdown for a worker, you can either use flower as described in the "Worker did not shut down properly" section above, or you can use this command:

```
python manage.py shutdown_celery_worker_by_hostname <worker name>
```

The **:raw-html-m2r:`<worker name>`** parameter will be one of the values you get from running `python manage.py show_celery_workers`.

Revoke celery tasks

To revoke specific tasks, issue this command, passing the fully-qualified python task names as args:

```
python manage.py revoke_celery_tasks <task name 1> <task name 2> ...
```

This command revokes all active, reserved, and scheduled tasks (from all workers that are online) matching any of the given fully-qualified task names. It's best to leave this command running for a bit in order to get them all, as it will keep polling for tasks to revoke, and there are likely to be tasks in the message queue which haven't been received by the worker yet. This command can be useful, for example, if there is a huge backlog of export tasks that are stale and are just causing the queue to be backed up. For example, to revoke all export tasks, keep this running for a bit (every time a task is revoked it will be printed to the console):

```
python manage.py revoke_celery_tasks corehq.apps.export.tasks.populate_export_download_task
```

You can pass more than one task to this command, and you can stop it any time with Ctrl+C.

Purge queue

NOTE: This should almost never be necessary in production and can be more useful during local development. In production it's better to target specific tasks to revoke using the revoke_celery_tasks command described above. In case you do need to do this in production, It's best to stop the worker that's consuming from that queue first:

```
sudo supervisorctl stop <...>
```

and then restart it after purging:

```
sudo supervisorctl start <...>
```

To purge all messages in a rabbitmq message queue:

```
celery -A corehq purge -Q queue_1,queue_2
```

We use rabbitmqctl to inspect RabbitMQ. All rabbitmqctl commands must be run as the root user on the machine hosting RabbitMQ.

Locally you can use sudo to run these commands, but in a production environment you'll need to switch to the root user first.

The **:raw-html-m2r:`<vhost name>`** parameter is commcarehq in our production environments. Locally you might have this set to /, but you can check it with the list virtual hosts command.

List Virtual Hosts

```
rabbitmqctl list_vhosts
```

List number of messages in each queue

This command lists the number of messages in each queue (i.e., the number of tasks in each celery queue) that are either ready to be delivered to a consumer or have been delivered but have not been acknowledged yet.

```
rabbitmqctl list_queues -p <vhost name> name messages
```

9.4 Specialized Howtos

Below docs contain how to instructions for special operations.

9.4.1 White label an existing CommCare HQ instance

Describes how to have two URLs point to the same CommCare HQ environment and serve separate branding for each.

Customization possible

- Login page can be customized per host with default: `CUSTOM_LANDING_TEMPLATE` in public.yml
- CommCare HQ name can be customized per host with default: `COMMCARE_HQ_NAME` in public.yml
- CommCare name can be customized per host with default: `COMMCARE_NAME` in public.yml
- Error pages can be customized by creating a new branch in the repository defined by `commcarehq_errors_repository`. To reference the new branch specify a new folder and the branch in `reach_errors_home` and `reach_commcare_errors_branch`

Not supported

- Emails will come from the same addresses
- A user's account will be shared between both URLs
- You cannot limit a domain to only one URL

9.4.2 Configure a firewall on the servers

In situations where the servers are not behind a spearate firewall it is necessary to run a firewall service on the each server to lock down access to unprotected services.

If you are unsure whether this is an issue or not you can try to access some of the services directly (replace <commcare.mysite.com> with the address of your CommCare instance):

Celery Flower: https://<commcare.mysite.com>:5555

Elasticsearch: https://<commcare.mysite.com>:9200

If either of those URLs work then you need a firewall.

Configuration

This setup uses ufw.

In order to configure the firewall you need to determine what network interfaces your servers have. There must be two interfaces, one for secure traffic and one for insecure traffic.

The public IP address of the proxy server should be connected to one of these interfaces.

In the example below `eth0` is the private interface and `eth1` is the public:

- Server1:
  - eth0: 172.16.23.1

  - eth1: 213.55.85.200

- Server2:
  - eth0: 172.16.23.2

To configure the firewall for this setup add the following to your environment's `public.yml` file:

```
ufw_private_interface: eth0
```

Now run the following command: *(before you run this make sure you have access to your servers in case you get locked out)*

```
$ commcare-cloud <env> ansible-playbook deploy_common.yml --tags ufw-proxy,ufw
```

This will apply the following rules:

proxy

```
allow 443 in (https)
allow 80 in (http)
allow 22 in (ssh)
allow 60000:61000 in (mosh)
allow all in on <ufw_private_interface>
block all in
```

all other servers

```
allow all in on <ufw_private_interface>
block all in
```

9.4.3 Adding a new machine to a cluster

#. Add the machine to inventory.ini #.

   Update the local known hosts

   ```
   $ cchq <env> update-local-known-hosts
   ```

1. For proxy, webworkers, celery, pillowtop run the following (this is the only step that modifies machines other than the new one):

   ```
   # Register with the shared dir host whitelist
   # so the shared dir host allows the new machine to mount the shared dir
   cchq <env> ansible-playbook deploy_shared_dir.yml --limit shared_dir_host
   ```

2. Deploy stack

   ```
      cchq --control <env> deploy-stack --first-time --limit <new host>

   If it fails part way through for transient reasons (network issue, etc.) and␣
   ↪running again fails with SSH errors, that means it has already switched over from␣
   ↪the factory SSH setup to the standard SSH setup we use, and you can no longer use␣
   ↪--first-time. To resume, run the following instead
   ```

```
$ cchq --control <env> deploy-stack --skip-check --skip-tags=users --limit <new␣
↪host>  # Only run this to resume if the above fails part way through
```

3. For anything that has commcarehq code you then have to deploy.

   To make sure all the machines are running the same version of commcare otherwise the machine that was just provisioned has the latest, and everything else has an old version.

```
# this is important in case there was drift between the branch where new setup was␣
↪run and master
# it also makes sure that all machines have the same settings
$ cchq <env> update-config
$ cchq <env> deploy
```

4. If your change involves shuffling processes around in app-process.yml or otherwise requires updating supervisor confs on machines other than the new one, run

   This updates the supervisor configurations (text files) on the relevant machine so that supervisor knows what processes should be running.

```
$ cchq <env> update-supervisor-confs
```

9.4.4 Performance benchmarking

Some useful tooling has been built for evaluating the performance of CommCare HQ.

Dimagi uses Locust for performance benchmarking. The code Dimagi uses is available at https://github.com/dimagi/commcare-perf . See the README for installation instructions, and further documentation on how to test with form submissions that mimic those of your project.

9.4.5 Migrating zookeeper to a new node.

This doc will explain how to migrate a running zookeeper to a new node without any downtime. First we will run ZooKeeper in replicated mode and shut down the primary server after that the secondary server will take over.

1. Add the new node in zookeeper group and run deploy-stack .. code-block:

```
cchq <env> deploy-stack --limit=<zookeeper node>
```

2. Edit conf `/etc/zookeeper/conf/zoo.cfg` on both the nodes

```
server.1=zookeeper node1 IP:2888:3888
server.2=zookeeper node2 IP:2888:3888
```

   Also edit `/var/lib/zookeeper/myid` to give each one an ID. (for ex 1 and 2) and restart the zookeeper.

3. Check if replication is working on both zookeeper node.

```
$ echo dump | nc localhost 2181 | grep brokers
```

   You should see same output on both the server.

4. Remove the old zookeeper node from zookeeper group and run deploy-stack again to update all the kafka server to connect with new zookeeper node.

```
$ cchq <env> deploy-stack --limit=<All kafka nodes> --tags=kafka,zookeeper
```

5. Shutdown the old zookeeper node.
6. Remove the old node entry from `/etc/zookeeper/conf/zoo.cfg` on new zookeeper nodes and restart the zookeeper service.

9.4.6 Add a new celery machine into existing cluster

Setup the new node

```
diff environments/<env>/inventory.ini
+ [celeryN]
+ <node ip>
```

```
diff environments/<env>/app_process.yml
+   'celery11':
+    reminder_case_update_queue:
+      pooling: gevent
+      concurrency: <int>
+      num_workers: <int>
```

Configure

1. Configure Shared Directory

```
commcare-cloud <env> ap deploy_shared_dir.yml --tags=nfs --limit=shared_dir_host
```

1. Deploy new node

```
commcare-cloud <env> deploy-stack --limit=celeryN
```

Update Configs

```
commcare-cloud <env> update-config
```

Deploy code

```
cchq <env> deploy
```

Update supervisor config

```
cchq <env> update-supervisor-confs
```

9.4.7 Add a new CouchDB node to an existing cluster

Setup the new node and add it to the cluster

1. Update inventory

```
+ [couchN]
+ <node ip>

  [couchdb2]
  ...
+ couchN
```

1. Deploy new node

```
commcare-cloud <env> deploy_stack.yml --limit=couchN
```

1. Add node to cluster

```
$ commcare-cloud <env> aps --tags=add_couch_nodes --limit=couchdb2
```

Migrate database shards to the new node

1. Create a plan

   e.g. 4 nodes with 3 copies of each shard (couchD is the new node)

   ```
   # myplan.yml

   target_allocation:
     - couchA,couchB,couchC,couchD:3
   ```

2. Create new shard plan

   ```
   $ commcare-cloud <env> migrate-couchdb myplan.yml plan
   ```

3. Compare the plan to the current setup

   ```
      $ commcare-cloud <env> migrate-couchdb myplan.yml describe

   Check that the new cluster layout is what you want. If not adjust
   your plan file and try again.
   ```

4. Create migrate (copy data to new node)

   This will shut down all the nodes in the cluster so make sure you have initiated downtime prior to this step.

```
   $ commcare-cloud <env> migrate-couchdb myplan.yml migrate

Alternatively, if you can be confident that the plan keeps
more than half of the copies of each given shard in place
(i.e. moves less than half of the copies of each given shard)
then you can use the ``--no-stop`` flag,
and the migration will be done with no downtime.
```

5. Commit the changes

   This will update the DB docs to tell Couch about the new shard allocation.

```
$ commcare-cloud <env> migrate-couchdb myplan.yml commit
```

6. Verify

```
$ commcare-cloud <env> migrate-couchdb myplan.yml describe
```

7. Redeploy Proxy

```
$ commcare-cloud <env> ansible-playbook deploy_couchdb2.yml --tags=proxy
```

Cleanup

After confirming that all is well we can remove old shards:

```
$ commcare-cloud <env> migrate-couchdb myplan.yml clean
```

9.4.8 Configuring VirtualBox for testing CommCare HQ

Step 1: Download and Install VirtualBox

Follow the instructions for your host machine operating system found at the VirtualBox Downloads page.

Step 2: Download, Install, and Configure Ubuntu

1. Download the latest Ubuntu 22.04 Server Edition ISO.
2. Configure VirtualBox
   - Open VirtualBox and create a new Virtual Machine.
   - Provide the VM with at least 16GB RAM, and a 40GB Disk, as per the minimum requirements for a monolith.
   - Once the VM is created, click Settings -> System -> Processor. Increase the number of processors to the maximum you can
   - Boot the VM and select the Ubuntu ISO you downloaded in the previous step
   - Follow the Ubuntu installation prompts, ensuring you install the OpenSSH server. The other defaults should all be left as-is unless you have specific requirements.

Step 3: Configuring VirtualBox Networking

There are two options for configuring the VirtualBox network.

Before following these instructions it is wise to read and understand the `NAT` and `Bridged` sections of this VirtualBox networking explained article. The rest of this section assumes knoweldge of that article.

NAT

This is the easiest, but will prevent you from accessing some CommCare features like `formplayer`.

Under the VM's network settings click "Advanced" then "Port Forwarding". Add the following rules:

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
SSH	TCP	127.0.0.1	2222		22
HTTPS	TCP	127.0.0.1	8083		443

With these settings:

- SSH into your server with:

```
$ ssh username@localhost -P 2222
```

- Access CommCare HQ from a browser at: .. code-block:

```
   https://localhost:8083
**Note**\ : the ``https`` part is important as redirects will not work using this␣
↪method.
```

Bridged

In this mode, the virtual machine will get its own IP address from the router that the host is connected to. This will allow the VM to be accessed from outside of the host.

Prerequisites

Bridged mode requires a few things from the host:

- A wired network connection, or a wireless connection that allows bridged connections (many wireless network cards and wireless gateways do not allow this)
- Ideally, an ability to give specific MAC addresses a static IP address from the network router. Otherwise the VM's IP address might change on reboot.
- An ability to edit the host's `host` file (`/etc/hosts` on unix machines)

Setting up Bridged mode:

- Under the VM's network settings, set the network adapter to be attached to the `Bridged Adapter`. Select the network device on the host that is connected to the router (i.e. your wireless or wired card).
- For some wireless gateways which require a password, you might need to set the MAC address of the to the MAC address of the host. This may sometimes work to get a new IP address, but some wireless gateways will only give a single IP per MAC.
- If you have access to the router, set it up to give the VM's MAC address in the settings with a static IP
- Boot your VM. If the settings are correct, the machine should boot and be given an IP address. Verify what the IP address is with: .. code-block:: bash

  $ ip addr

- On the host, edit the `/etc/hosts` file: .. code-block:: bash

  $ sudo nano /etc/hosts

  and add the following line to the end:

```
{ip address of the guest} monolith.commcarehq.test
```

With these settings:

- SSH into your server with:

```
$ ssh username@{ip address of the guest}
```

- Access CommCare HQ from a browser at: .. code-block:

```
https://monolith.commcarehq.test
```

Set up Sentry for error logs has infomration on how you can send metrics to sentry.io account from your CommCare HQ instance. commcare-cloud also allows you to self host Sentry instance. The below guide describes how to do this.

9.4.9 Setup Sentry self hosting

Running sentry on self hosting requires following minimuim services to be running

- Postgresql
- Redis
- Kafka
- zookeeper
- Snuba
- Clickhouse server
- Sentry

How to setup

1. Add a server to `sentry` and give it a var `kafka_broker_id`
2. Setup following vars in `public.yml` .. code-block:

```
sentry_dbuser:
sentry_dbpassword:
sentry_database:
sentry_system_secret_key:
clickhouse_data_dir:
default_sentry_from_email:
```

3. Add the database and host detail inside `postgresql.yml`
4. Run following command .. code-block:

```
$ cchq <env> ap deploy_sentry.yml --limit=<sentry_hostname>
```

5. After the command is finished, ssh into sentry server, activate the sentry virtualenv and create a superuser for it. .. code-block:

```
(sentry_app) root@MUMGCCWCDPRDRDV01:/home/sentry/sentry_app# sentry --config /home/
↪sentry/config/ createuser
Email: test@test.com
Password:
Repeat for confirmation:
Should this user be a superuser? [y/N]: y
User created: test@test.com
Added to organization: sentry
```

6. Login to the sentry UI and now this admin account can be used to manage the sentry.

9.4.10 Tips and Tricks

The following list outlines some useful things that may help when doing specific tasks. Before using them you should make sure you understand what it is doing and when it is appropriate to use.

Update localsettings in a specific release

Limitations

This only works for CommCare processes and not for Formplayer.

Usage scenario

Testing configuration changes without impacting running processes

Setup

Create a new release:

```
commcare-cloud <env> deploy commcare --private [--limit ...]
```

Note down the release folder location: `/home/cchq/www/<env>/releases/YYYY-MM-DD_HH.MM`

Update configuration in that release only

```
commcare-cloud <env> update-config --limit [LIMIT] -e code_home=[RELEASE FOLDER]
```

This will override the default value of the `code_home` variable which normally points to the `current` release.

Choosing a value for `LIMIT`:

- If you did not use `--limit`, set `LIMIT` to `django_manage`
- If you did use `--limit`, set `LIMIT` to the same value as used before.

9.4.11 How To Rebuild a CommCare HQ environment

These steps delete *all* CommCare data in your environment.

In practice, you will likely *only* need this to delete test environments. We strongly discourage using any of these of steps on production data. Please fully understand this before proceeding as this will permenantly delete all of your data.

Prior to Wiping Data

1. Ensure CommCare services are in a healthy state. If you observe any issues, see the Troubleshooting section below.

   ```
   $ cchq <env_name> django-manage check_services
   ```

2. If planning to migrate data, deploy CommCare from a specific revision

   ```
   $ cchq <env_name> deploy commcare --commcare-rev=<commit-hash>
   ```

   Note: You should have been given a commit hash that matches the revision of CommCare used to generate your exported data, and it is critical that this same CommCare revision is used to rebuild the new environment, and load data in. Please request a commit hash if you were not provided one.

3. Stop CommCare services to prevent background processes from writing to databases.

   ```
   $ cchq <env_name> downtime start
   # Choose option to kill any running processes when prompted
   ```

How To Wipe Persistent Data

These steps are intended to be run in the sequence given below, so you shouldn't proceed to next step until the prior step is completed.

1. Ensure CommCare services are stopped to prevent background processes from writing to databases.

```
$ cchq <env_name> service commcare status
```

2. Add "wipe_environment_enabled: True" to *public.yml* file.

3. Wipe BlobDB, Elasticsearch, and Couch using management commands.

```
$ cchq <env_name> django-manage wipe_blobdb --commit
$ cchq <env_name> django-manage wipe_es --commit
$ cchq <env_name> django-manage delete_couch_dbs --commit
```

4. Wipe PostgreSQL data (restart first to kill any existing connections)

```
$ cchq <env_name> service postgresql restart
$ cchq <env_name> ap wipe_postgres.yml
```

5. Clear the Redis cache data

```
$ cchq <env_name> django-manage flush_caches
```

6. Wipe Kafka topics

```
$ cchq <env_name> ap wipe_kafka.yml
```

7. Remove the "wipe_environment_enabled: True" line in your *public.yml* file.

Rebuilding environment

1. Recreate all databases

```
$ cchq <env_name> ap deploy_db.yml --skip-check
```

2. Run migrations for fresh install

```
$ cchq <env_name> ap migrate_on_fresh_install.yml -e CCHQ_IS_FRESH_INSTALL=1
```

3. Create kafka topics

```
$ cchq <env_name> django-manage create_kafka_topics
```

Warning: If you are migrating a project to a new environment, return to the steps outlined in *4. Import the data to the new environment*. Do not start services back up until you have finished loading data into your new environment.

Start new environment

Note: The following steps should only be run if you are **not** planning to migrate a project from an existing environment.

1. End downtime (you will encounter a prompt that says no record of downtime was found, continue anyway as this starts services up).

```
$ cchq <env_name> downtime end
```

2. Recreate a superuser (where you substitute your address in place of "you@your.domain").

```
$ cchq <env_name> django-manage make_superuser you@your.domain
```

Troubleshooting

Issues with check_services

- Kafka: No Brokers Available - Try resetting Zookeeper by performing the following steps:

```
$ cchq monolith service kafka stop
NOTE: The following paths may vary if you've specified different paths for ``kafka_
↪data_dir`` and ``zookeeper_data_dir``
$ rm -rf /var/lib/zookeeper/*
$ rm -rf /opt/data/kafka/data/*
$ cchq monolith service kafka restart
```

9.5 Settings in `public.yml`

The following are settings found in the `public.yml` file. Many are passed on to CommCare HQ in its `localsettings.py` file:

9.5.1 Email addresses

daily_deploy_email:
Notifications are emailed to this address when a deploy completes.

root_email:
Used as the email address when requesting SSL certificates from LetsEncrypt. Proxy server notifications are also emailed to this address.

server_email:
Used as the "from" or "reply to" address by CommCare HQ for:

- Bug reports from CommCare users
- Notifications of requests for project spaces or organizations
- Notifications from the `sync_prepare_couchdb_multi` management command

server_admin_email:
All CommCare HQ web service (Django) administrator email notifications are sent to this address. (It is used as the address for Django ADMINS and MANAGERS.)

default_from_email:
Used as the "from" address for:

- All emails sent via the Celery email queue
- "Dimagi Finance" and "Dimagi Accounting" on automated accounting reports, sales requests, subscription reminders, invoices, weekly digests of subscriptions, and pro-bono applications
- The `send_email` management command

return_path_email:
The email account for receiving bounced emails. This needs to use an IMAP4+SSL compliant service. It is tested with GMail.

Used by the `process_bounced_emails` management command.

support_email:
This is the address given to users to reach out to for support in situations where they may have queries, for example, in password reset emails, project space transfers, the 404 page, among others.

In non-Dimagi environments this address is given as the email for Support in CommCare apps.

probono_support_email:
The address given for Support in pro-bono application submissions.

accounts_email:
The email account to which generated invoices and weekly digests are sent, and to which subscription reminders are CCed. It is also the contact address given for subscription changes.

data_email:
The address to which the monthly Global Impact Report is sent.

subscription_change_email:
Notifications of subscription changes are sent to this address.

internal_subscription_change_email:
Notifications of internal changes to subscriptions are sent to this address.

billing_email:
Enterprise Plan and Annual Plan requests are sent to this address. It is also given as the contact address for signing up for new subscriptions.

invoicing_contact_email:
The contact email address given on invoices, and notifications regarding payment methods.

growth_email:
Subscription downgrade and pause notifications are sent to and BCCed to this address.

saas_ops_email:
Unused

saas_reporting_email:
The "Credits On HQ" report is sent to this address for the "production", "india" and "swiss" Dimagi environments.

master_list_email:
This address is sent a list of self-started projects which have incomplete info and over 200 form submissions.

sales_email:
Given as the contact address if users need more OData feeds, more Report Builder reports, or messaging/SMS features.

privacy_email:
Given as a contact address in the End User License Agreement.

feedback_email:
Feedback for features is sent to this address.

eula_change_email:
When a user changes a custom End User License Agreement or data-sharing properties for their domain, a notification is sent to this address.

contact_email:
Unused

soft_assert_email:
Soft asserts in the source code that that are called with `send_to_ops` send to this email address.

new_domain_email:
This address is notified when a user requests a project space or organization.

9.6 Ports Required for CommCare HQ

Below are the list of ports for various services required for running CommCare HQ.

Process	Port	Internal Access	External Access	Allow in Iptables? Monolith Env	Allow in Iptables? Non-Monolith OR O	Comments
SSH	22	yes	Restricted IPaddress	yes	yes	
Nginx https	443	•	yes	yes	yes	
Nginx http	80	•	yes	yes	yes	
Monolith Commcare	9010	yes	no	no	depends	:sub:`routed via nginx `
Formplayer	8181	yes	no	no	depends	Accessible to private network
Kafka	9092	yes	no	no	depends	Accessible to private network
Zookeeper	2181	yes	no	no	depends	Accessible to private network
Redis	6379	yes	no	no	depends	Accessible to private network
PostgreSQL PgBouncer	5432 6432	yes	no	no	depends	Accessible to private network
RabbitMQ	5672	yes	no	no	depends	Accessible to private network
ElasticSearch ES Cluster	9200 9300	yes	no	no	depends	Accessible to private network
CouchDB	5984 4369	yes	no	no	depends	Accessible to private network
Celery port			no	no		
Mail/SMTP ports	25 465 587		yes	no		

CHAPTER

TEN

ABOUT THIS CHANGELOG

Below are the list of changes (the newest first) to `commcare-cloud` and CommCareHQ that need to be applied on your environment to keep it up to date.

10.1 Changelog

10.1.1 2024-04-30 Copy Supply Point data to Invitation model Location Field

Copy data from `supply_point` field to `location` field in `Invitation` model to prevent errors with future migrations to the `users_invitation` table.

This operation is required for all environments regardless of feature usage.

10.1.2 2024-02-13 SQL Repeat Record Migration addendum

Optional: Copy repeat records with deleted repeaters from Couch to SQL

10.1.3 2024-03-29 Upgrade To Node 20

Node.js 16.x LTS is reaching its end of life 15th June 2024, so node and npm must be upgraded on all machines.

10.1.4 2024-02-13 SQL Repeat Record Migration

Prepare for and migrate Repeat Records from Couch to SQL.

10.1.5 2024-01-14 Elasticsearch upgrade from 2.4.6 to 5.6.16

Upgrade to Elasticsearch 5.

CommCare HQ releases after March 1, 2024 will not support Elasticsearch 2.x. So we strongly recommend applying this change before then.

10.1.6 2023-10-25 Reindex All Indices For Elasticsearch Upgrade

Reindex Elasticsearch Indices for upcoming ES Upgrade.

10.1.7 2023-09-12 update-to-python-3.9.18

Installs python 3.9.18 and build a new virutalenv for CommCare HQ

10.1.8 2023-06-14 Prepare project spaces for Case List Explorer report release

We have created a management command to assist with syncing data to the Case Search Index so that legacy projects may access data in a new report that will be made generally available.

10.1.9 2023-04-10 upgrade-redis-to-v7

This change upgrade Redis from 6.x to 7.0 version. As part of our ongoing effort to keep CommCare HQ up to date with the latest tools and libraries we have updated Redis from version 6.2 to version 7.0.

10.1.10 2023-03-23 Upgrade to Ubuntu 22.04

On April 1, 2023, the operating system that commcare-cloud has supported for the last 4-5 years, Ubuntu 18.04 "Bionic Beaver", will reach its end of life for support from its maintainer.

We are requiring that all CommCare HQ instances be upgraded to Ubuntu 22.04 "Jammy Jellyfish" by July 1, 2023, in order to continue using commcare-cloud. **Failure to update by this date will result in increasingly unexpected behavior possibly including downtime and data loss.**

Schedule maintenance for a time well before July 1, 2023, during which you should expect a downtime of up to many hours. We also recommend making contingency plans in case issues arise during the first attempt at applying this maintenance. For example, you may want to set an early date in April, with backup dates in May and June in case issues arise during the first attempt.

10.1.11 2023-03-09 Update Node to v16

Node.js 14.x LTS is reaching its end of life 30th April 2023, so node and npm must be upgraded on all machines.

10.1.12 2023-01-10 Configure Java 17 for Formplayer

In preparation for the release of Formplayer Java 17 version, we have shipped a few infrastructure changes through Commcare Cloud. At this point, the ask to those maintaining CommCare instances is to follow the steps below to configure Java 17 for Formplayer.

10.1.13 2023-01-10 Install Dart Sass as npm global dependency in preparation of Bootstrap upgrade

Install Dart Sass as global NPM library.

10.1.14 2022-12-20 Dropping support for 3.6

Commcare-cloud will no longer attempt to support Python 3.6. We strongly encourage you to follow the steps below to upgrade to Python 3.10 if you have not done so already.

10.1.15 2022-12-13 remove-unused-elastic-settings

CommCareHQ has two settings (`CASE_ES_DROP_FORM_FIELDS` and `ES_XFORM_DISABLE_ALL`) that were created several years ago for tuning Elasticsearch in a specific environment which no longer exists. This change removes support for those settings and the application logic that existed for them. We do not expect this to impact any self-hosted environments, but it's prudent to confirm.

10.1.16 2022-12-13 Populate a new field for Data Forwarders

Populate a new field for all Data Forwarders (repeaters).

This is optional, but is recommended to do for all environments where Data Forwarders are being used. A migration will be added in the future, which will automatically apply these changes during a code deploy if they have not yet been applied. If there are many Data Forwarders, it will slow down the deploy process.

10.1.17 2022-11-28 New script added for virtualenv activation

Instructions to resolve this issue connecting to remote machines: "/home//commcare-cloud/control/activate_venv.sh: No such file or directory"

10.1.18 2022-11-11 Backfill SMS event data for API performance

Backfill data in the `sms_smssubevent` table to improve the performance of the 'messaging-event' API.

This operation is only required if your environment is using SMS or Email workflows. Furthermore, an automatic migration will be added to CommCare in the future which will apply these changes during a code deploy if they have not already been applied.

10.1.19 2022-11-08 kafka-upgrade-to-3.2.3

Run command to upgrade Kafka version to 3.2.3 Update Scala version to recommended version 2.13

10.1.20 2022-11-08 Install Elasticsearch Phonetic Analysis Plugin

The Elasticsearch 'case search' index now requires the phonetic analysis plugin to be install in Elasticsearch.

10.1.21 2022-11-07 Upgrade to Python 3.10

Follow these steps to install and use Python 3.10 on your control machine(s) by December 19th, 2022.

10.1.22 2022-11-01 postgres-v14-upgrade

Postgres upgrade is recommended to be upgraded in two steps,upgrade from v10 to v13 and then to v14. This change upgrade PostgreSQL from 10 to 13 and then to 14 version. As part of our ongoing effort to keep CommCare HQ up to date with the latest tools and libraries we have updated PostgreSQL from version 10 to version 14.

10.1.23 2022-10-30 RabbitMQ upgrade to 3.10.7 version

This change upgrade RabbitMQ 3.10.7 version. The current installed RabbitMQ version 3.8.5 is already End of Life. So, we strongly recommend applying this change.

10.1.24 2022-09-15 Update Prometheus variable

If using prometheus, run `update-config` to apply changes to prometheus environment variable.

10.1.25 2022-09-15 upgrade-kafka_3.2.0

Run command to upgrade Kafka version which 2.6.1 to 3.2.0 Update Scala version to recommended version 2.13

10.1.26 2022-09-14 bootstrap-userrole-auditing

Run command to create bootstrap audit records for existing user role records.

10.1.27 2022-09-05 upgrade-zookeeper_3.7.1

Run command to upgrade Zookeeper version which 3.2.0 to 3.7.1

10.1.28 2022-08-23 upgrade-redis

This change upgrade Redis from 4.0.8 to 6.x version. As part of our ongoing effort to keep CommCare HQ up to date with the latest tools and libraries we have updated Redis from version 4.0.8 to version 6.2.

10.1.29 2022-08-19 upgrade-ansible

Run command to uninstall older ansible version which 2.9.26 and install 4.2.0

10.1.30 2022-08-11 historical-auditcare-data-migration

Instructions regarding Migrating Historical Auditcare data to SQL.

10.1.31 2022-06-17 Remove report_cases and report_xforms indices

CommCareHQ has two elasticsearch indices "report_cases" and "report_xforms" that were created many years ago for some custom reports, but which haven't been used in some time. This change deletes those indices and the change processors used to populate them. We do not expect this to impact any self-hosted environments, but it's prudent to confirm.

10.1.32 2022-06-03 Postgres upgrade from 9.6 to 10

This change upgrade PostgreSQL from 9.6 to 10 version. As part of our ongoing effort to keep CommCare HQ up to date with the latest tools and libraries we have updated PostgreSQL from version 9.6 to version 10.

10.1.33 2022-05-25 update-supervisor-confs

Run command to update shell runner scripts for django and celery.

10.1.34 2022-04-19 Update PgBouncer configuration to support multiple processes

This change updates the PgBouncer role to support multiple processes on a single machine, so that more CPU cores can be used (PgBouncer is single-threaded and uses only one CPU core by default).

10.1.35 2022-04-11 Upgrade Node.js to 14.19.1 and npm to 7.24.2

Node.js 12.x LTS is reaching its end of life 30th April 2022, so node and npm must be upgraded on all machines.

10.1.36 2021-11-02 Upgrade CommCare HQ to Python 3.9

Install Python 3.9 and build a new virtualenv for CommCare HQ.

10.1.37 2021-06-09 Migrate forms & cases from Couch to SQL

A series of management commands must be run to check for and migrate domains' forms & cases from Couch to SQL.

10.1.38 2021-03-23 Optionally configure auditcare db

Some enterprise deployments of CommCare use a backend feature called "auditcare" as part of their audit logging strategy. Auditcare is enabled by default, so it is active unless you went out of your way to disable it when you configured CommCare. Historically it has used CouchDB as its data backend, but it is being switched to use PostgreSQL instead. If you care about this feature then you may want to carefully consider this change log before your next commcare deploy; otherwise you can ignore.

10.1.39 2021-03-16 Update Formplayer request mapping

A small change to the nginx request matching for Formplayer requests that will prevent an additional / from being prepended to the Formplayer requests.

10.1.40 2021-01-11 Dropping support for Python 2

Python 3.6 is supported and now the preferred version to use. In anticipation of dropping Python 2 support, an error will be displayed when running commands with Python 2. Instructions for upgrading to Python 3 are provided in the error message. An option is provided to temporarily revert to Python 2 (see update steps for details).

Python 2 support will end on 2021-03-04.

10.1.41 2021-01-08 Install new `apt` requirements on machines running commcarehq code

We will be adding SAML 2.0 Single Sign On (SSO) support over the next few months which requires installing new `apt` packages as dependencies of python requirements.

10.1.42 2020-11-16 update-letsencrypt-to-alternate-chain

On January 11 2021, Let's Encrypt will change its default certificate chain from using the legacy Identrust root certificate, to its own modern ISRG root certificate. In order to maintain backwards compatibility with existing mobile devices it is necessary to keep using the Identrust certificate chain.

10.1.43 2020-10-22 Update PostgreSQL monit configurations to be version specific

Update PostgreSQL monit configuration files to be version specific and use `systemctl`

10.1.44 2020-10-14 Run command to update Supervisor configurations

Run management command to remove unused errand-boy processes.

10.1.45 2020-07-23 Run command to clear sensitive information from Django sessions

It is stronly recommended to sanitize legacy Django sessions after upgrading to Django 2.2.

10.1.46 2020-07-09 Switch to new package manager

To stay up to date with javascript package security alerts, we need to make sure we are using a javascript package manager that is supported by Github.

10.1.47 2020-04-16 Update git to the latest version

Due to a high-severity security advisory on the popular version control software program `git`, observing security best practices dictates upgrading to one of the git versions designated as "Patched" such as 2.26.1.

10.1.48 2020-02-28 Update `deploy` CommandArgs

We are adding support for deploying from a specific commit hash or tag, replacing the deploy command's `commcare-branch` argument with a more general `commcare-rev` argument.

10.1.49 2020-02-05 ES upgrade from 1.7.6 to 2.4.6

This change upgrade Elasticsearch from 1.7.6 to 2.4.6 version. CommCare HQ releases after April 2, 2020 will not continue to support Elasticsearch 1.7.6, so we strongly recommend applying this change before then.

10.1.50 2019-11-06 Update Formplayer Configuration

Some properties in the Formplayer configuration have changed names.

10.1.51 2019-08-30 Upgrade Sentry

The Sentry SDK used by CommCare HQ has been updated and along with it we have updated the configuration parameters.

10.1.52 2019-08-23 Add `deploy` Command

In order to provide a consistent user interface while making underlying changes, we are replacing the `commcare-cloud <env> fab deploy` command with a more concise `commcare-cloud <env> deploy` command.

10.1.53 2019-08-23 Removing support for Riak CS

We are removing support for deploying Riak CS clusters in commcare-cloud

---

10.1.54 2019-08-23 Fix python3 virutalenvs (Deprecated)

Update (2019-11-27): This fix is no longer necessary as it has been superceded by changes to the deploy script that make this change automatically if necessary.

This fixes a bug with how python3 virtualenvs were created by ansible. This fix needs to be applied to any machine which has a python3 virtualenv that was created by commcare-cloud.

The fix is also safe to run on all CommCare hosts.

10.1.55 2019-08-21 Move remaining management commands explicitly

This change requires editing `app-processes.yml` to add some of processes to the `management_comamnds` section

---

10.1.56 2019-08-20 Rename management commands explicitly

This change requires editing `app-processes.yml` to rename some of the processes in the `management_comamnds` section

---

10.1.57 2019-07-17 Define management commands explicitly

This change requires changing app-processes.yml to include a list of management comamnds to run

---

10.1.58 2018-07-18 Upgrade to Python 3

This change installs Python 3.6.8, builds a new virtualenv, and runs CommCare HQ in Python 3.

10.1.59 2019-05-13 Install Pango

This change installs pango and its dependencies for the weasyprint library which has been added as a requirement to commcare-hq for proper pdf printing of unicode fonts

---

10.1.60 2019-02-26 Fix to restart nginx after every letsencrypt cert auto-renewal

Previously you had to manually restart nginx every time letsencrypt auto-renewed, which was about every two months. We believed we had fixed this with Restart nginx after every letsencrypt cert auto-renewal, but there was an error in our setup at that time that made it not work as intended.

10.1.61 2019-04-05 Update RabbitMQ logging configuration

This change updates the RabbitMQ logging configuration to change the log level from `info` to `warning`.

10.1.62 2019-02-27 Remove celery results backend from localsettings

Upgrading to celery 4.x requires removing the dependency on django-celery, which means that its results backend will no longer be available. This removes the django-celery backend as the default from localsettings, so the results backend can be specified by commcare-hq settings instead.

10.1.63 2019-02-26 Split pgbouncer vars from postgresql vars

This change extracts a new role from the existing postgresql role for installing and configuring pgbouncer.

As a result of this change the `postgresql.yml` environment configuration file needs to be changed to split out the postgresql vars from the pgbouncer vars.

10.1.64 2019-02-27 Only monitor specific RabbitMQ queues

Datadog RabbitMQ monitoring restricts the number of queues it can monitor to 200. To avoid hitting this limit on large scale deployments we limit the queues being monitored to only the primary queues.

10.1.65 2019-02-22 Update supervisor confs to invoke celery directly

Upgrading to celery 4.x requires removing the dependency on django-celery, which means that the celery management command becomes unavailable. This prepares for that by invoking the celery command directly.

10.1.66 2019-02-22 Separate celery datadog http check

This adds a specific http check for the celery check (serverup.txt?only=celery) to datadog. Environments that are not relying on datadog for monitoring can ignore this change.

10.1.67 2019-02-11 Add tag to datadog http checks

This change adds "check_type" tag to the http_check datadog integration. This change applies only to envs using datadog for monitoring.

10.1.68 2019-02-11 Java upgrade for formplayer

Previously, Formplayer was running on Java 7. This change updates us to Java 8 for formplayer.

10.1.69 2019-02-01 Generalize load case from fixture feature

Previously loading a case from a fixture required the fixture to be an attribute. This change allows using non-attributes from the fixture.

10.1.70 2019-01-16 Fix encrypted temp directory permissions

This is a followup to Added encrypted temporary directory in which we introduced an encrypted directory for temp files. In its original implementation, this file was owned by root, and processes were unable to write to it.

This changes the directory to be owned by cchq, allowing our processes to write to the file.

10.1.71 2019-01-02 Restart nginx after every letsencrypt cert auto-renewal

Update 2019-02-26: There was a bug in this fix and it has been superceded by Fix to restart nginx after every letsencrypt cert auto-renewal.

Previously you had to manually restart nginx every time letsencrypt auto-renewed, which was about every two months.

10.1.72 2018-12-15 Blob Metadata Migration - part 2

Form submission attachment metadata is being consolidated in the blob metadata table in SQL. This migration consists of a series of commands that will consolidate the data in your environment.

10.1.73 2018-09-24 Blob Metadata Migration - part 1

Blob metadata needs to be migrated from CouchDB to SQL. This migration consists of a series of commands that will move the data in your environment.

10.1.74 2018-11-26 Reorganize pillows

Pillows read changes from kafka and do various processing such as sending them to elasticsearch, transforming into a UCR table row etc. A doc for same change is read multiple times for each processor, since there are separte pillows for each processor. This is inefficient, so we have combined multiple processors that apply for a given document type (also called `KAFKA_TOPIC`) such as form/case/user under one pillow. For e.g. A new single `case-pillow` pillow replaces various old pillows that process case changes such as `CaseToElasticsearchPillow`, `CaseSearchToElasticsearchPillow`, `ReportCaseToElasticsearchPillow`, and `kafka-ucr-main` etc.

10.1.75 2018-11-20 New Case Importer Celery Queue

Importing cases is often a time-sensitive task, and prolonged backlogs are very visible to users. It will be useful to have a separate queue specifically for case imports, to improve visibility into backups as well as typical runtimes. Additionally, this is a first step towards allocating resources specifically for case imports, should that become necessary.

10.1.76 2018-08-16 Support multiple Kafka brokers

Large scale deployments of CommCare require scaling out Kafka brokers to support the high traffic volume (as well as for high availability). Up until now CommCare has only supported a single broker.

10.1.77 2018-08-16 Add queue for analytics tasks

Tasks for analytics reporting have been separated into a new analytics celery queue.

10.1.78 2018-07-25 Update Supervisor

Ubuntu 14.04 `apt-get install supervisor` installs supervisor 3.0b. We occasionally have issues that could be related to supervisor, such as processes not stopping correctly. To rule it out as a possible cause, we decided it was better to be on a later version of supervisor, and one that's not in beta.

10.1.79 2018-07-13 Update supervisor service definitions

There are several CommCare specific processes that are defined in supervisor configuration files. This change decouples the process definitions from code.

10.1.80 2018-06-11 Added encrypted temporary directory

Some of the CommCare processes make use of temporary files to store client data (such as data exports) so in order to keep that data protected we have modified the setup to use an encrypted temporary directory.
