MAGENTO CLOUD CERTIFICATION

STUDY GUIDE

JOSEPH MAXWELL

INTRODUCTION

You downloaded the most comprehensive study guide for the Magento 2 Certified Professional Cloud Developer certification (testing your knowledge of Magento Commerce 2.3). This covers the exam which was released in early 2019.

This test covers the A-Z of releasing and then maintaining a website on the Magento Cloud platform. We might be tempted to say, "it's hosting—what is so hard about that?" and then not put effort into studying for the test. I can personally attest to the fact that this is a well-written and difficult test.

Our goal is that this guide serves two purposes:

1. For those who are new to Magento, this guide serves as an overview of managing Magento projects. Neither the guide nor the test is technical in that they require knowledge of code. Rather, the guide helps you understand the areas that practical experience is necessary.
2. For those who are experienced in Magento, this guide will help fill in the voids to ensure a thorough understanding of how it works.

How can you achieve this certification?

I would love to say that all you have to do is read the study guide, and you will pass. That might be true for some but not the majority. To pass the test, I suggest:

- Ensure you have a Cloud sandbox, at least.

- Study this guide. I know, it's not like reading a novel.
- Establish consistency in your study: every day from 7:30am - 8:30am, before you begin your work for the day.
- Obtain practical experience. There are some practical experience notes throughout the study guide. Do those. Then, make up some of your own.
- Once you feel comfortable, take the SwiftOtter practice test (https://swiftotter.com/certifications). It will be a reality check: "am I ready to pass the test?"

While I have done my best in providing quality and helpful content (while also being succinct), it is your responsibility to study, practice, and prepare. With that, go and do it!

Acknowledgments

I would like to extend heartfelt gratitude to the following individuals who contributed by edits or comments to the development of the study guide. These people graciously donated their time to help others out. If you see them at a conference or on Twitter, please thank them.

- Billy Gilbert (Adobe)
- Barny Shergold (Vaimo)
- Matthew Beane (Zilker)
- Luc, Navarr, Andre and the other awesome developers at MedioType
- Juan Pablo Giménez Giaimo
- Leo Gumbo (We are JH)

- Anton Pavelko
- Nikola Lardev
- James Cowie (Shero)
- Manish Jain
- Jose carlos Filho Velasco
- Saulo DSCF
- Derrik Nyomo

All the best!
Joseph Maxwell

CONTENTS

1 COMMERCE CLOUD FUNDAMENTALS

13% of the test

SWIFT OTTER SOLUTIONS

1.1 DESCRIBE THE FEATURES AND FUNCTIONS OF MAGENTO COMMERCE CLOUD

What is Magento Cloud?

Magento Cloud is a hosting platform built on AWS to host Commerce projects. This represents an entire toolset built on Platform.sh and includes many additional value-add features. This system represents the most qualified hosting platform as Magento manages the code and the environment.

Cloud platform overview and features

There are two editions of Magento Cloud: Starter and Pro. Magento Starter and Pro have almost the same features, with the exception that Pro has:

- A dedicated hosting environment (i.e. dedicated AWS instances)
- Three server setups for fault tolerance

Further Reading:

- Magento Commerce Cloud Guide

- B2B module (like on-premise Magento installations, you need to install this manually via composer)

A slightly different path for code to get to production (Integration > Staging > Production with 8 available active environments)

Starter

This edition of Magento Cloud utilizes the `master` branch for production. This means that once you merge code from the `staging` branch, `master` will be deployed to production.

Note that while code flows from integration to staging to production, data flows in the opposite order: production back to staging, back to integration. This is because production should be the single source of truth for all information stored in the database.

The `master` and `staging` branches should have no changes directly made to them (although it is still possible). The workflow for Starter sites is `master` back to the integration branches, where work happens:

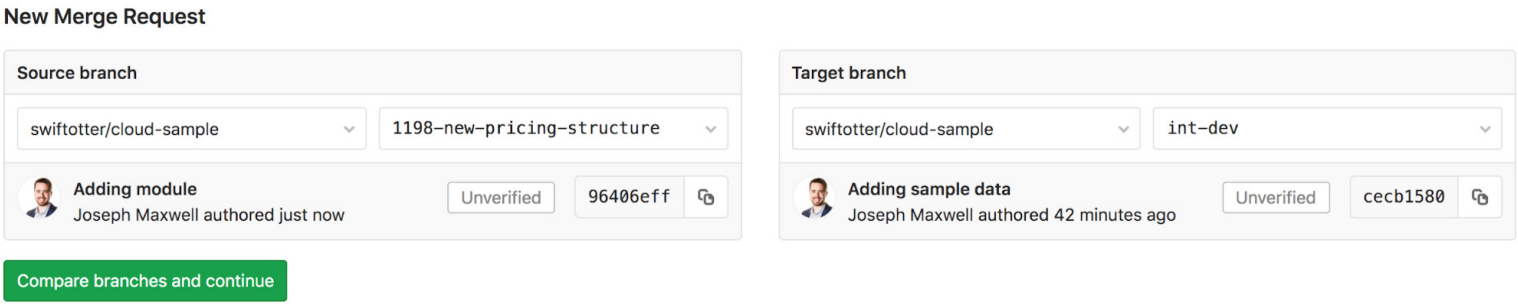
- `master` should have no changes made directly to it.
- `staging` is checked out from `master`. Code pushed to this branch is then pushed on to the `master` branch. This environment closely replicates the primary environment with a read-only file system. **This is the branch from which you create integration branches.**
- Next, are integration branches. This is where work happens. While you can have two active integration environments (branches, which are activated in the Cloud control panel), you can push as many branches as you wish.

Development workflow:

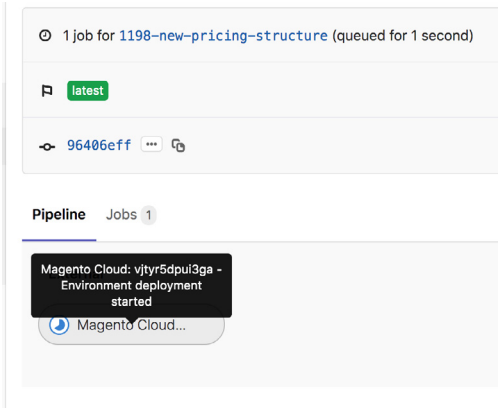
- Check out `staging`.
- Pull the latest code for `staging` from `platform` remote.

- Create a new integration environment branch. This is named whatever you wish. The only "magic" branch names are `staging` and `master`.
- If desired, create a new feature branch. Depending on how your team handles release cycles, you could have an integration branch/environment for the release of updates, with the other integration branch/environment ready for immediate bug fixes.
- Make changes on a new feature branch (if necessary).
- Merge code back into the integration branch and push. This will deploy code to your integration environment if it is activated.
- Merge code from the integration branch into `staging`. This will deploy code to the `staging` environment.
- Upon final testing, and (hopefully) approval from the merchant, code is pushed to `master`. This releases updates to production.

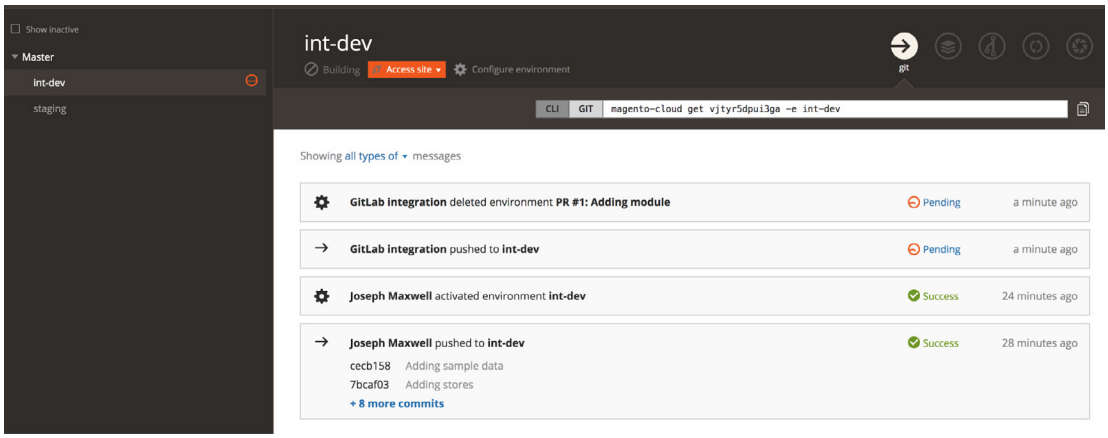
Creating a merge request for adding a new module:



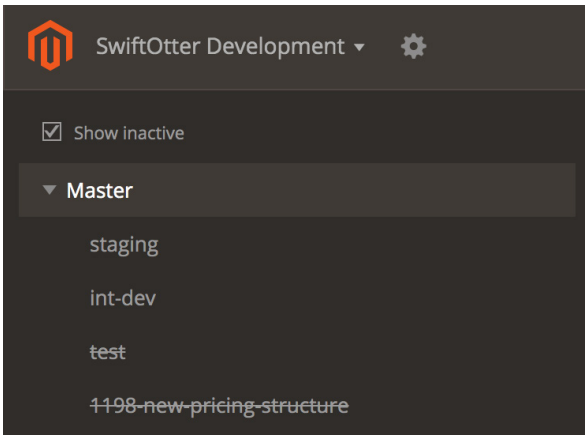
Once merged:



Magento Cloud:

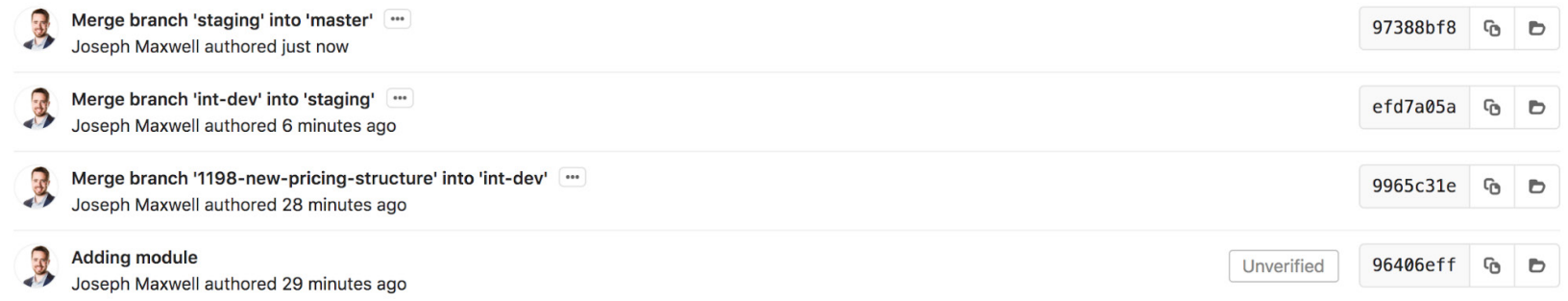


Upon performing the above steps, your environment should look roughly like this:



Note:

- There is no "hierarchy" of branches. `staging` is promoted, because it is, well, special.
- Remember that for Pro environments, you must branch from integration.
- If you create or install any new modules, make sure to enable those with `bin/magento module:enable MODULE/NAME` on your local system first (obviously, you won't do this in a Cloud environment). You should have `app/etc/config.php` included in your Git repository. One other thing to note is that any modules that are not explicitly disabled in `app/etc/config.php` are automatically enabled.
- Whenever a new environment is activated, data is automatically copied from the environment it is being branched from. Keep any data regulations in mind as customer data will be copied.
- It is up to the lead developer to determine which branches should be activated as an environment (anything that is not struck out above is an active environment).
- You can easily use GitHub, GitLab, or BitBucket instead of using Magento git hosting. The benefit of utilizing Magento's git environment is you see logs in your Terminal window when you `git push` instead of having to go to Magento Cloud to see logs. One advantage of using GitLab or a similar service is you can protect specific branches to prevent accidental pushes.



Practical experience:

It is highly advised that you have a Cloud sandbox. If you work for a partner or are a Community Insider, you should have access to one.

Initialize your environment:

- Clone repository to your system. Click "git" (with a right arrow) to see the `git clone` path.
    - Note that you can use the `magento-cloud` commands for this (for example `magento-cloud checkout <PROJECT ID>`, although they might seem less familiar).
- If necessary/desired, configure the project locally on your system (using the `master` branch for Starter projects or `integration` for Pro projects).
- Push up to the appropriate branch.
- SSH into production and configure:
    - Create an admin user.
- If you need to initialize a new/separate environment and Magento is NOT installed, a deploy will attempt to automatically install Magento.
- If you need to reset an environment:
    - SSH into the branch's environment.
    - Run `bin/magento setup:uninstall`.
    - Clear all writeable files (`rm -rf ~/*`).
    - Flush Redis (`redis-cli -h redis.internal FLUSHALL`)
    - Push an empty commit to the branch.
    - Install Magento/import database.
    - `echo $MAGENTO_CLOUD_RELATIONSHIPS | base64 --d | json_pp` (to see database and redis credentials)

Pro

The Pro version of Magento Cloud is more structured than the Starter version.

In Starter, you had a free-floating `master`, `staging` and `integration` branches, where `staging` is the primary branch from which to create new branches. In Pro, the `integration` branch is where you create branches to develop new features. Note that in GitLab terms, `master`, `staging` and `integration` would all be protected.

Child branches from `integration` are then used to activate an environment.

Once you merge changes from `staging` to the `master` branch, it is recommended to also push to the global master branch.

Otherwise, development on Pro and Starter are very similar.

Further Reading:

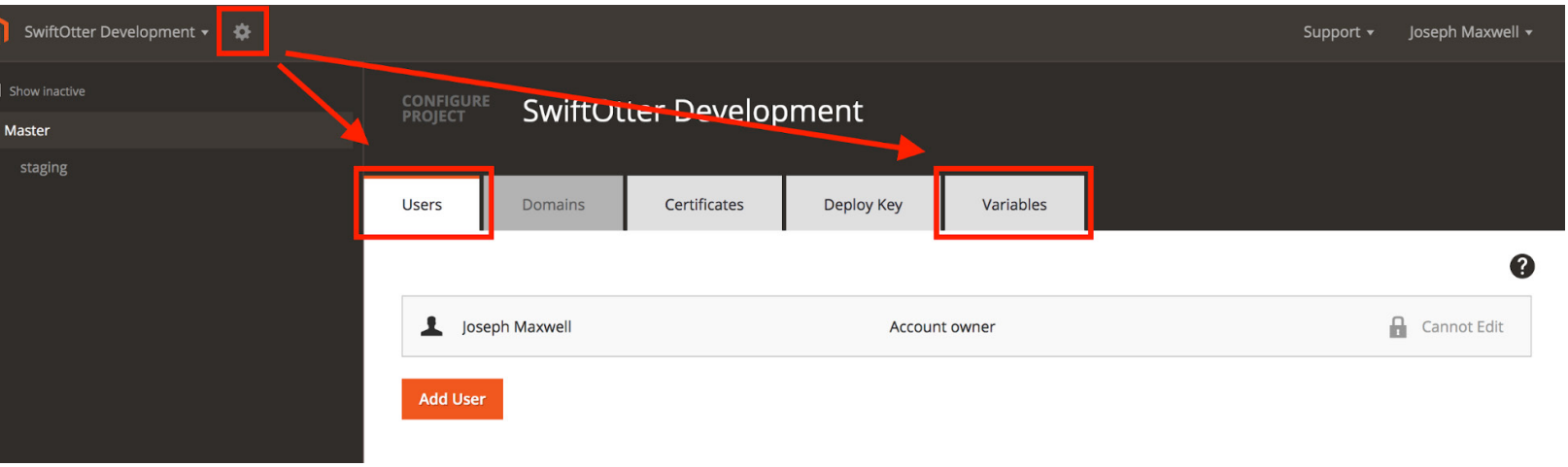
- Magento Commerce Cloud Architecture
- Starter Architecture
- Pro Architecture
- Pro Develop and Deploy Workflow

1.2 DETERMINE HOW TO LOCATE SETTINGS WITH CLOUD ADMIN UI

Instead of just reading through the following, please use it as a guide to your own practical experience of the Admin UI.

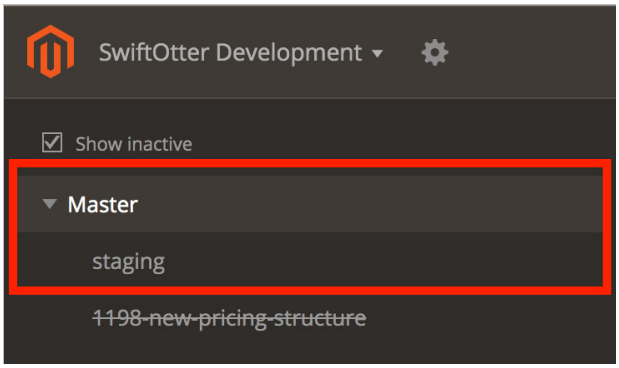
Locate project settings, user management, and project variables pages

Project settings

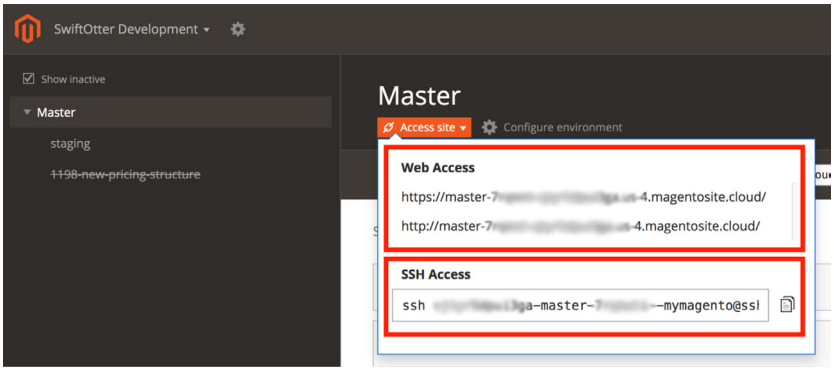


Locate environments, access links, and logs

Environments:

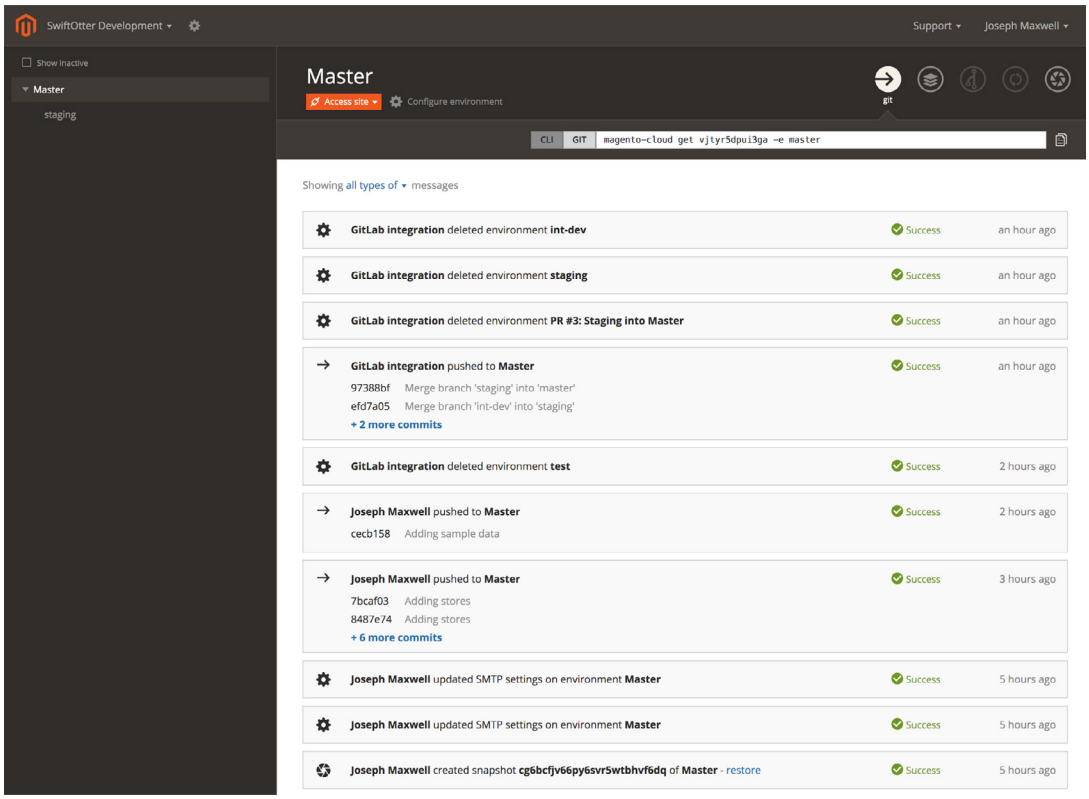


Access links

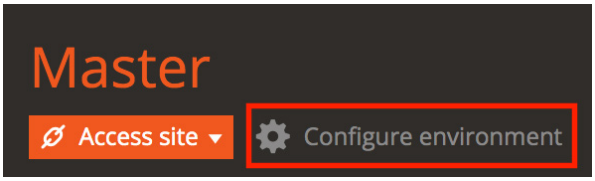


Logs

Logs are stored in `/var/log` (for Starter and Pro Integration environments) and accessed via SSH. For Pro Staging and Production, these are in `/var/log/platform/<PROJECT ID>` (remember that for Pro accounts, you will need to log into all three nodes). Basic environment actions are available in the environment control panel:



Locate environment settings

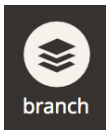


Inside of each environment's configuration, you can:

- Change settings:
  - Update the environment's status (disable it, for example).
  - Enable outgoing emails (this is disabled by default).
  - Indexing by search engines.
  - Configure HTTP access control (ideally used on staging to prevent search engines and unauthorized access).

Additionally, there are 5 very important buttons at the top of each branch:

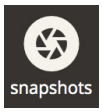
This provides the commands and git remote details for loading this project locally to your system.



If this button is enabled, you are able to create a new branch (environment) from this current branch (environment). Note that the DevDocs and the Admin UI use branch and environment interchangeably. The only possible difference is that an environment is a branch that is now publicly accessible via the web (environment is enabled).

Merges the branch to its master. This means `integration` to `staging` and `staging` to `master`. It goes without saying that this is not available on the `master` branch (but, I did say it).

Syncs the code and/or data and files. This can also be done via the `magento-cloud` CLI.



Creates a backup.

1.3 DEMONSTRATE THE ABILITY TO MANAGE USERS

Add SSH key

Go to Account settings > Account Settings > SSH Keys. You can also add this via the CLI: `magento-cloud ssh-key:add`.

Practical experience: do it. Ensure your SSH key is properly configured in your a account.

Add users to a project and manage their roles

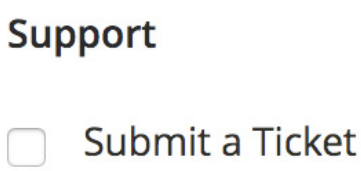
Click the gear icon next to your project > Users > Add User. The user should get a welcome email from Magento with instructions on how to get started. You can also use the `magento-cloud` CLI to add users.

You can allow users access per project and then per environment (inside the project). Each user can be designated as a "Super User/Admin", which allows them to do anything in any environment. If this is not enabled, you can

designate their access per environment. When you change their permissions, you must redeploy (if nothing else, using `git commit --allow-empty`).

- **Admin**: as you probably have guessed, this person can do just about anything (they are the only ones that can access SSH).
- **Contributor**: can merge and branch from an environment. You can allow contributors to SSH into the environment with using `ssh: contributor` in the `.magento.app.yaml` file.
- **Reader**: like reading books from the library but more boring (as Magento Cloud is not a historical novel).

Note that providing access to support tickets is a completely different mechanism than providing access to configure environments. This happens on the magento.com `portal`. By default, the account owner (as seen in the top ribbon settings gear icon > Users > Account owner) only has permission to create a support ticket. However, in their Magento portal, they can share access to "Submit a Ticket":



Further Reading:

- Create and manage users

1.4 DETERMINE THE DIFFERENCE BETWEEN MAGENTO CLOUD PLANS

Starter plan vs. Pro differences

As discussed above, merchants on Cloud Pro are isolated into their own environments. This prevents collateral damage from one merchant's "go-live" taking down another merchant.

The Starter plans allow for 3 testing environments (integration branches). Pro contains 1 integration environment with capacity for 4 more active branches.

Code in Cloud Pro navigates from Active Branch > Integration > Staging > Production.

Code in Cloud Starter navigates from Active Branch (Integration) > Staging > Production.

Cloud Pro also includes the B2B module.

Further Reading:

- Magento Commerce Cloud architecture

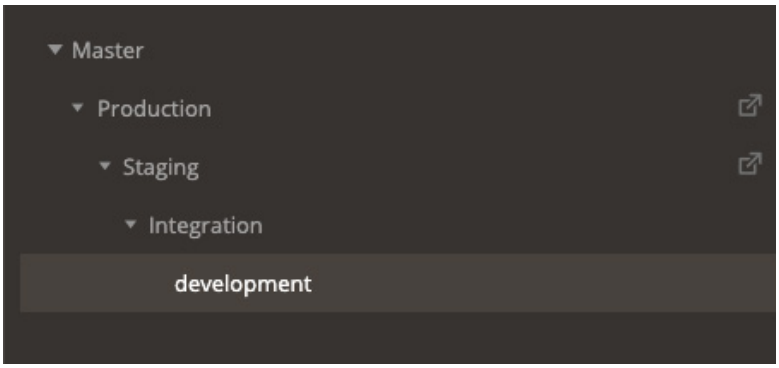
1.5 DETERMINE HOW DIFFERENT ENVIRONMENT TYPES OPERATE

Determine differences in environment types: Integration, staging, production

- Integration:
  - Up to two (Starter) or four (Pro) environments. Pro also has an official `integration` branch/environment that feeds up to `staging` branch/environment.
  - Pro: this branch (environment) serves as the source for all new branches. Once code is merged into the `integration` branch, it is merged up through staging and production and global master. As such, any fixes or even hotfixes should come from integration (technically, you can push to staging and production directly if you need to get a patch into production quickly).
- Staging:
  - One environment that is based on the `staging` branch in the git repository. This environment is designed to closely replicate production. All services available on production (like Fastly, New Relic, and Blackfire) are available on staging. This environment is read-only. Code should first be deployed to an integration branch before going to staging or production unless there is an emergency and you can then push directly to staging or production.
- Production:
  - One environment
  - Read-only

Pro Notes:

You should only branch from the integration branch (it is technically possible to create from other branches but not recommended).



Above is a screenshot of a Pro environment. Note that the `development` branch was branched from integration (`integration` in the git repository).

2 LOCAL ENVIRONMENT

9% of the test

SWIFT OTTER SOLUTIONS

2.1 DEMONSTRATE ABILITY TO SET UP LOCAL DEVELOPMENT

Software you need to have locally for developing a Magento Cloud project

- LAMP stack (Valet+, Vagrant)
- OR Docker via `ece-tools` (this is quickly becoming the preferred method).

You can use the Magento Docker (from `ece-tools`) or your own environment. While the Cloud Composer meta-package includes additional tools, the base Magento Commerce application is present. This means you can use your own system, like Vagrant, to run Cloud projects.

The benefit of Docker is that it closely mirrors the environment that Magento will run in production. For example, you can run `./vendor/bin/ece-tools docker:build --mode="production"` to stand up a test environment locally that is read-only. This helps to vet out problems that happen should modules try to save files to protected directories. Docker is becoming intelligent so as to replicate the configuration specified by `.magento.app.yaml` which is a huge plus.

One thing to note is that Pro plans use a custom `auto_increment_increment` value (3). This is because the production environment has three nodes. Additionally, you must never set the value of a primary key. This could cause overlap/inconsistent data in the production environment.

Further Reading:

Launch Docker

2.2 GIVEN A SCENARIO, DEMONSTRATE ABILITY TO USE THE MAGENTO-CLOUD CLI TOOL

Install Magento-cloud CLI.

- Install the CLI via one-line installer (`curl -sS https://accounts.magento.cloud/cli/installer | php`).
- Include path in environment variables.

To see all available commands, run `magento-cloud list`.

The `magento-cloud` CLI tool is pretty much used for general project configuration and information. This is not to be confused with the `ece-tools` binary (found in `vendor/bin/ece-tools` inside a Cloud project) that performs build, cron, docker and other actions on a specific Cloud project.

Further Reading:

Magento Cloud CLI reference

Magento Cloud CLI (local)

Retrieve project info

```
magento-cloud project:info
```

This command provides detailed information about the status of the instance (including whether or not the subscription is in effect).

Manage project and environments

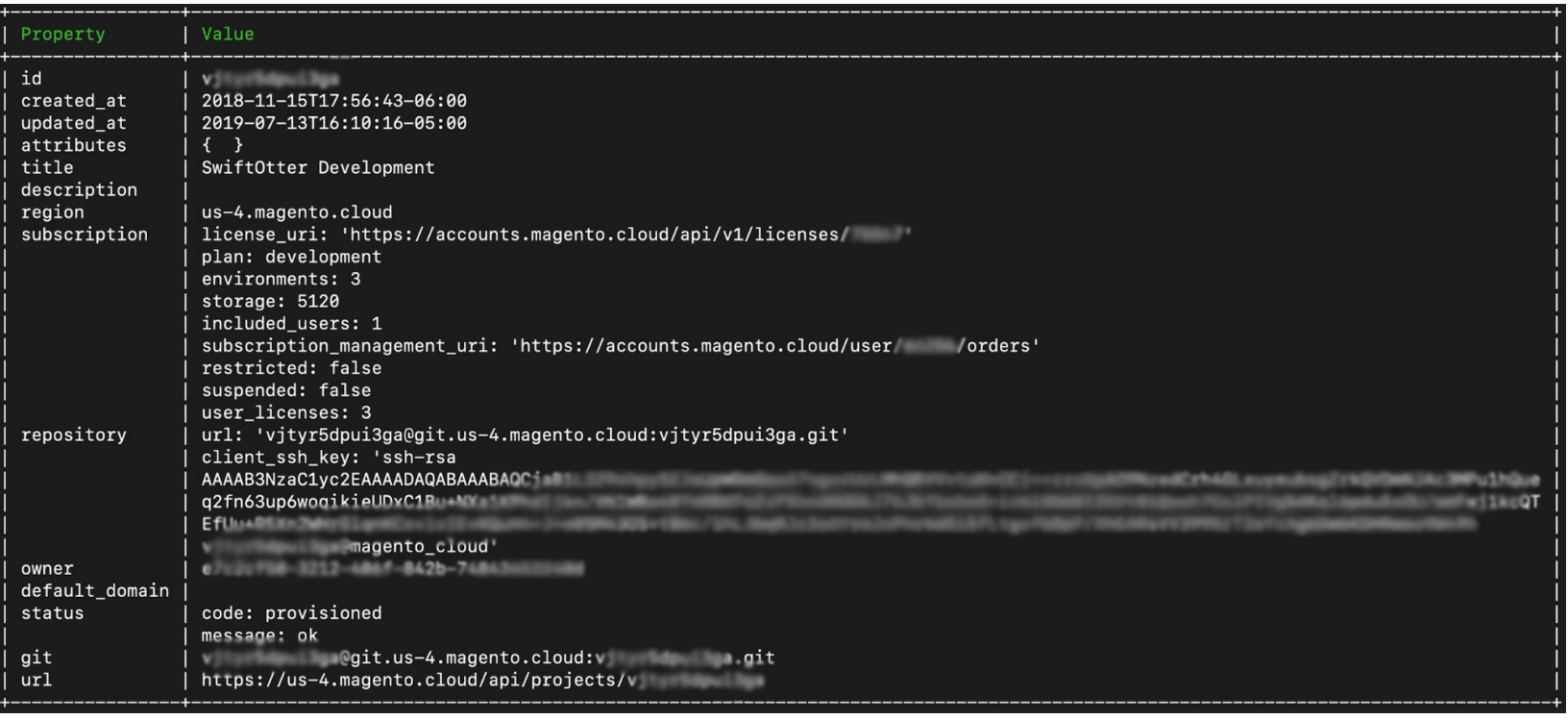
project:clear-build-cache **Clear a project's build cache**

As discussed below in "Build the project locally," part of the battery of tests to ensure a successful delivery is to test that the build works as expected. This is especially helpful if you have a custom build hook (in `.magento.app.yaml`) and need to verify that this works as you need it to. This command clears the cache created from these builds.

project:get **Clone a project locally**

This associates the current directory with a new Magento Cloud project.

project:info **Read or set properties for a project**



See above for a screenshot.

project:list (projects, pro) **Get a list of all active projects**

Shows all projects for a user.

project:set-remote **Set the remote project for the current Git repository**

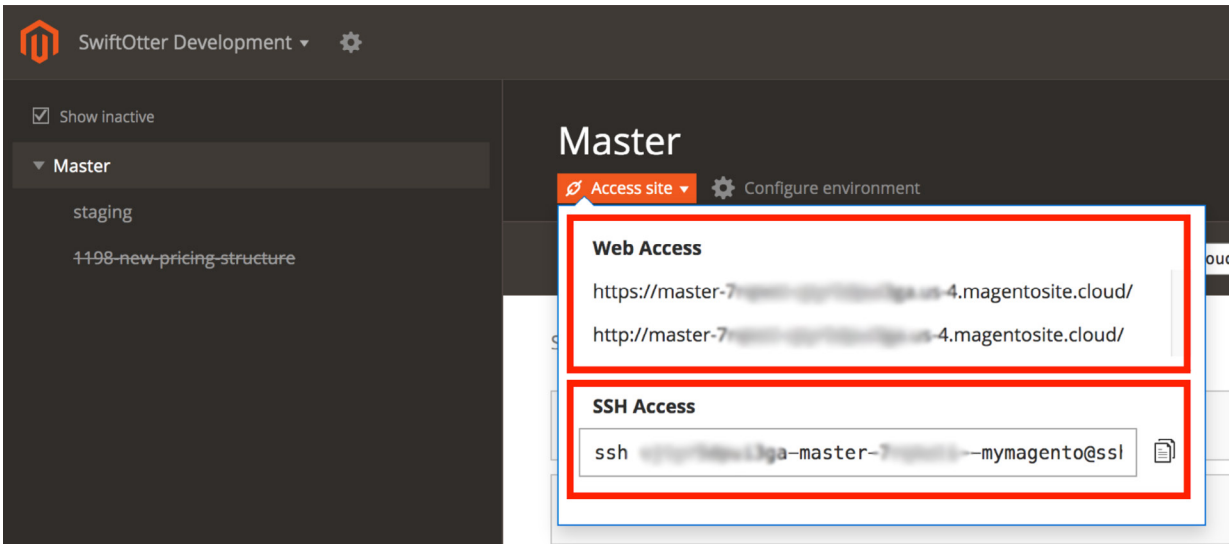
Further Reading:

Magento Cloud CLI reference

Connect to database and SSH

The Cloud dashboard provides a very easy way to access SSH. Remember, you need to configure your SSH key in your account before you'll be able to access SSH. To do this, you must go to account settings (in the upper right-hand corner) > Account Settings > SSH Keys.

All you have to do is copy in the SSH details to your bash window:



Access the database

To access the database use these details:

- Host: `database.internal`
- Username: `user`
- Password: (empty)
- Database: `main`

You can also use the `magento-cloud db:sql` command to load these connection details.

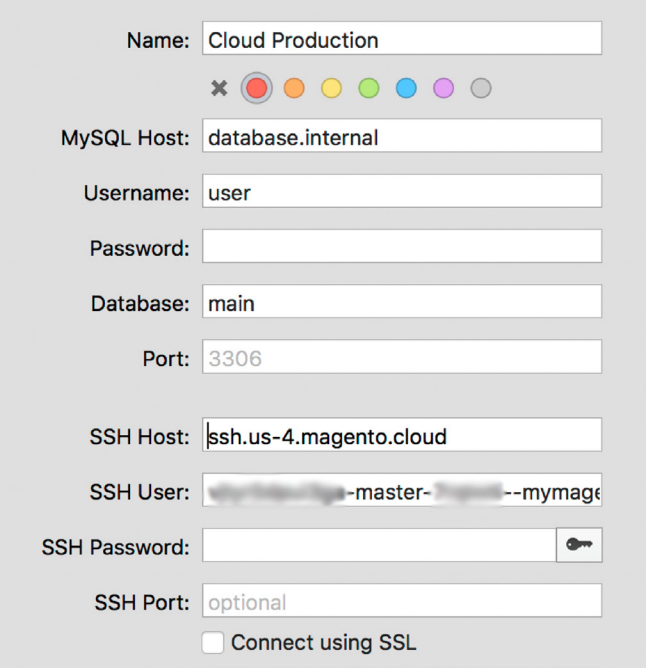
On Pro, Staging, and Production, databases need to be connected with specific usernames and passwords as well as a different host:

- Host: `127.0.0.1`
- Username: `<project id>` (suffix with _stg for staging)
- Password: <db_password>
- Database: `<project id>` (suffix with _stg for staging)

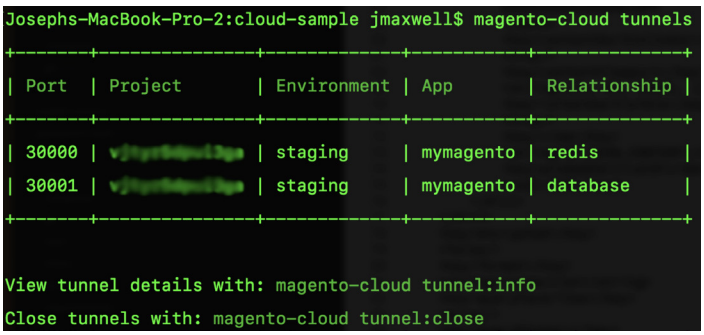
DB Password can be accessed by SSH to node 1 and retrieved from app/etc/env.php.

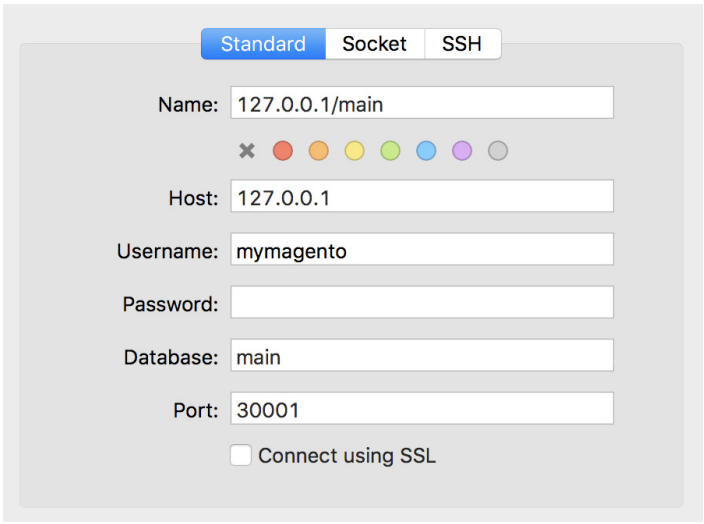
For the sake of simplicity, you can combine the user and the host in the SSH Host field.

You can also use a tunneled connection with the `magento-cloud` client to connect to your database:



```
magento-cloud tunnel:open
```





Notes:

Set up MySQL service

Build the project locally

Before going live, or to troubleshoot a build, you can execute the build process locally. This allows you to intercept potential problems before they are pushed to the git repository (which kicks off a build process). If you do this, **please make sure you delete all files that this process generates.** If they end up in your git repository, it could cause problems for the build process.

Here is the command to begin the build:

```
magento-cloud local:build
```

Once complete, run:

```
magento-cloud local:clean
```

For what it's worth, the console app is built in Symfony and imports the Platform.sh PHP client through Composer.

Further Reading:

Test build your code locally before pushing

3 CLOUD CONFIGURATION

13% of the test

SWIFT OTTER SOLUTIONS

3.1 DETERMINE HOW TO CONFIGURE CLOUD

How to configure different redirects in this file, which types of redirects should not be configured here

`.magento/routes.yaml` provides a codified means to manage redirects on the Magento site. Gone are the days of managing .htaccess files. This redirect system is quite powerful.

For example, if you wish to enforce a naked domain name (non-www), use this `.magento/routes.yaml` file:

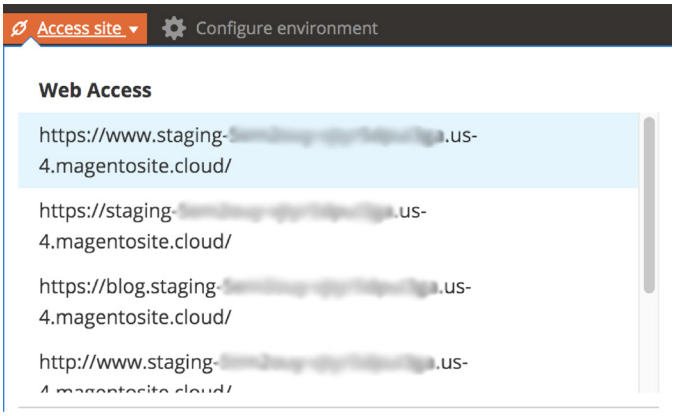
```
"http://www.{default}/":
    type: redirect
    to: "http://{default}/"

"http://{default}/":
    type: upstream
    upstream: "mymagento:http"
```

In the above, you will notice the references to `{default}`. This is substituted for the domain name for the environment. You can also use `{all}`. This is useful when your Magento website services multiple domains (https://swiftotter.com and https://swiftotter-training.com, for example).

You might wonder "what is the purpose of type: upstream?" This is used to serve content to the name listed in .magento.app.yaml (the left side of the colon should always match, see here). If you wish to serve the website from a

sub-domain name, you would add that. You can use wildcards for matching. For example, if you would like to configure a sub-domain for a Magento website scope, this is how you would configure the web server part of the update.

Notice that any time you add a new domain into `.magento/routes.yaml`, that domain appears in the Access site menu:



Practical experience:

- Set up `.magento/routes.yaml` in your project.
- Redirect www.[yourdomainname] to shop.[yourdomainname].
- Serve Magento from blog.[yourdomainname].

Redirects are very powerful in `.magento/routes.yaml`. You can also specify a partial redirect. This is helpful if you are able to redirect URL structures from an old site to a new site in a specified manner.

Further Reading:

Redirects

Configure routes

How to add these configurations to Staging or Production environments.

For Starter projects, the changes to `magento/routes.yml` are automatically applied when you merge code. However, for Pro, you must create a support ticket.

Magento On-Premises installation migration. How to migrate an existing Magento installation into Magento Cloud: Code base, database, media migration

To prepare an existing Magento project (i.e. load in/merge files from the Magento Cloud base project):

- Create Cloud configuration files (`.magento.app.yaml`, `magento-vars.php`, `.magento/services.yaml` and `.magento/routes.yaml`).
- Configure composer authentication credentials (via an environment variable or `auth.json`).
- Require the Cloud meta package with the same version as was previously installed (you also need to make sure `NonComposerComponentRegistration.php` is referenced in the files, but all new Magento projects already have this).

Once you have your code pushed to the git source for Cloud (either Magento's git repository or your hosted one elsewhere):

- Load in the database.
- Update the URLs to match the domain name.
- Set the encryption key in `app/etc/env.php` for each environment.
- Load in the media.

As mentioned in the docs, if you wish to prevent your `auth.json` file from being present in production or visible in a repository, utilize a project

(environment) variable, `env:COMPOSER_AUTH`. Configure this in the main project configuration (top menu gear icon) > Variables. You must ensure that this variable is used in builds.

Practical experience:

Load an existing project into Cloud. Don't take a merchant's website without their explicit permission as this could violate GDPR and other rules.

Further Reading:

Prepare your existing Magento Commerce install

Add Magento authentication keys

3.2 DETERMINE HOW TO CONFIGURE A PLANNED SERVICE

How to configure a service that is planned to be added to the environment

Services are enabled in `.magento/services.yaml` and then associated with your project in `.magento.app.yaml`.

Note that you are able to configure multiple entries for each service (i.e. two MySQL databases).

These services are available:

- `mysql` (default installed)
- `redis` (default installed)
- `elasticsearch`
- `rabbitmq`

```
# .magento/services.yaml
elasticsearch:
    type: elasticsearch:6.5
    disk: 256

rabbitmq:
    type: rabbitmq:3.5
    disk: 256
```

Note the disk space configuration. This allows you to increase the utilized space for each service. Once a service's disk space has been increased, it is not possible to reduce it. Also, remember that these updates do not affect staging or production on Pro unless you file a support ticket. The default storage amount per project is 5GB. You will receive an error if a service's disk size is < 256MB.

Then, you need to let the Magento environment know about these services:

```
# .magento.app.yaml

relationships:
    rabbitmq: "rabbitmq:rabbitmq"
    elasticsearch: "elasticsearch:elasticsearch"
```

To specify additional details about these services, see details in Objective 4.

When you tunnel into this environment, these services will also be exposed for you to access and debug. Renaming a service first deletes the old environment then creates the new one. As such, data in the old service is first deleted.

3.3 DEMONSTRATE ABILITY TO ADD TO YOUR ENVIRONMENT

Which configurations you can add to your environment and how to do it

There are four primary files used to configure environments:

- `.magento.app.yaml`: the configuration file for a Magento application.
- `.magento/services.yaml`: this enables and configures MySQL, Redis, ElasticSearch, and RabbitMQ.
- `.magento/routes.yaml`): this configures caching and redirects.
- `.magento.env.yaml` is used by all environments.

`.magento.app.yaml`

This is where the primary application configuration occurs. Please review the documentation if you have not done much with Magento Cloud in the past.

Key points:

- `access`: used to declare the minimum user role that has SSH access. This can be `admin`, `contributor`, or `viewer`.
- `crons`: this configures cron jobs to run (as you might guess).

- `hooks`: there are three sections, `build`, `deploy`, and `post_deploy`. This is discussed below, too.
- `mounts`: this configures new writable directories. If you wish to make a directory editable, add it here. If you wish to make that directory browsable on the frontend, add it into the `web` section, too.
    - By default, these are configured:
        - `var/`
        - `app/etc/` (so both `env.php` and `config.php` are writable)
        - `pub/media`
        - `pub/static`
- `name`: this must match the name specified `.magento/routes.yaml`.
- `relationships`: as described in Objective 4, this section makes services available to the Magento application.
- `dependencies`: this allows you to specify packages that you want included into the build and deploy environments. You can specify pages for `ruby` and `nodejs`.
- `runtime/extensions`: adds PHP extensions to the environment. Redis, ImageMagic, and Xsl are required to be present. However, to utilize the feature set in Cloud, also include the BlackFire and New Relic extensions.
- `variables`: environment variables can be set in one of four places:
    - `.magento.app.yaml`: creates global environment variables, specified in code, for all environments (as deployed).
    - `.magento.env.yaml`: creates environment variables for specific stages (`global`, `build`, `deploy` and `post-deploy`). This is also used to configure notifications and logs.
    - In the web admin, per environment
    - In the web admin, for the project
- `web`: see section below.

Note that you can also create a custom `php.ini` file to control PHP's configuration. This configuration is appended/merged with the existing Magento Cloud configuration. This is done by adding a `php.ini` file to your root project directory.

`web` section

This represents an incredibly powerful portion of Cloud configuration. While the documentation indicates that there are two areas to the `web` section, **there aren't**. The `document_root`, `passthru`, etc, are all deprecated and not used anymore. Instead, we will discuss the options found in `web/locations`.

- Each exposed location is a key inside of `locations`. If you look at the default configuration file, you will see three: `/`, `/media`, and `/static`. Note that the first one, `/`, allows for more files/locations through the `rules` node.
- `root`: this is the root directory that is served. In most cases this will be `pub`. In the event of using a custom PWA, this might be a different path.
- `passthru`: this indicates the root file for the website should the requested resource not be found. If the user requests `catalog/product/view/id/1981`, that path doesn't exist as a file on the system, so the `passthru` file will pick it up (ie Magento). For the `static` directory, this will be `/front-static.php`. For the `media` directory, this will be `/get.php`.
- `index`: which index files are accepted? For Magento, this will be `index.php`. For content directories, this might not be set.
- `scripts` and `allow`: I can't find documentation on this. However, it seems that `scripts` is `true` for applications and the reverse for static content. `allow` can be overridden for specific `rules` (see below) and likely refers to allowing static files.
- `rules`:
  - `allow`: whether or not to allow this regex pattern when it matches a static file. Note that for the main default application, `allow` is `false`,

and then is overridden for paths that match, to name a few: `.css, .js, .png, .gif, .jpg`.

- `passthru`: this maps the request to another path. Note the rule in `/static` to accommodate the static content signing signature:

```
^/static/version\d+/(?<resource>.*)$:
                    passthru: "/static/$resource"
```

Practical experience:

- Deploy a new service. Go to the new environment and run:

```
php -r 'print_r(json_decode(base64_decode($_ENV["MAGENTO_CLOUD_
RELATIONSHIPS"])));'
```

- Configure a new mount point and make it available to be browsed on the frontend.

Further Reading:

Application

Configure routes

Services

Build and deploy

What to configure in this file, on which environments these configurations are applied, how to add these configurations to environments where this file is not read

- `.magento.app.yaml` is utilized in integration, staging, and production environments for both Starter and Pro projects. If you enable ElasticSearch with `.magento/services.yaml`, configure the service relationships in `.magento.app.yaml`, and deploy to production on a Pro project, you may run into problems.
- `.magento/services.yaml` is not available in staging and production environments for Pro projects.
- `.magento.env.yaml` is used in all environments.

Further Reading:

Services

4 SERVICE CONFIGURATION

5% of the test

SWIFT OTTER SOLUTIONS

4.1 DEMONSTRATE ABILITY TO CREATE SERVICE CONFIGURATIONS

How to add system services: MySQL, Redis, Elasticsearch, RabbitMQ

Note that the `services.yaml` file and the services-affected portions of the `.magento.app.yaml` file only affect the integration environment on Pro Cloud accounts. To change services in staging or production, you must file a support ticket. However, this does not apply to Starter Cloud hosting.

The basic syntax for enabling a new service in `services.yaml` is:

```
elasticsearch:
    type: elasticsearch:6.5
    disk: 1024
```

The `type` contains the name of the service with the required version number. The `disk` entry is how large (in megabytes) that you wish for the disk allotment to be. This must be a minimum of 256MB. If you do not specify a big enough disk space, you will get some strange errors. For example with MySQL, that would be "MySQL server has gone away." That's a helpful error (not really).

Once you configure the services in `.magento/services.yaml`, you then need to map those services to useable "access points." This happens in `magento.app.yaml`:

```
relationships:
    …
     elasticsearch: "elasticsearch:elasticsearch"
```

One thing to note is that configurations have the `_merge` option. If `_merge` is enabled, the values specified in this section will be merged with those in `app/etc/env.php`. If a `*_CONFIGURATION` is used and `_merge` is not set, then the applicable section in `app/etc/env.php` will be overwritten with what is specified in the `*_CONFIGURATION` section.

For example, let's say you want to specify a custom database user. You can add that with:

```
# .magento.env.yaml
stage:
  deploy:
    DATABASE_CONFIGURATION:
      username: 'my_user'
```

If you commit/deploy the code above, your database connection will break because your `app/etc/env.php` will look like:

```
// app/etc/env.php
'db' =>
  array (
    'connection' =>
    array (
      'default' =>
      array (
        'username' => 'my_user'
      )
    ),
  ),
```

Magento will not be able to connect to the database. The resolution is to utilize the `_merge` flag:

```
# .magento.env.yaml
stage:
  deploy:
    DATABASE_CONFIGURATION:
      username: 'my_user'
      _merge: true
```

With this, your database will still break (unless you have configured `my_user` as a valid user), but all other configuration details will be present:

```
// app/etc/env.php
'db' =>
  array (
    'connection' =>
    array (
      'default' =>
      array (
        'username' => 'my_user', // changed
        'host' => 'database.internal',
        'dbname' => 'main',
        'password' => '',
      )
    ),
  ),
```

The most useful time for this is to specify an external database or connection (like to ElasticSearch).

The same holds true for specifying a `table_prefix` or using any of the other `CONFIGURATION` options.

MySQL

Cloud can have multiple databases. To do this, you must specify a unique endpoint (user) for each.

You can create multiple users (called `endpoints`). By default, the user (endpoint) is `mysql`. However, in our experience, with the default configuration, you can use any value (or leave empty) the username.

Practical experience:

- Configure two users to access the MySQL database. One should have all privileges, the other should just allow `SELECT` queries.

Further Reading:

Set up MySQL service

Redis

To use Redis, make sure you configure the `runtime/extensions/redis` and `relationships/redis` in `.magento.app.yaml`. Otherwise, configuration for this is pretty basic.

If you wish to access the Redis databases, you can use the `magento-cloud tunnel:open` feature to establish a local, tunneled connection. Then, you can connect to it using the `redis-cli` on your system.

ElasticSearch

Again, similar to the above. However, you are able to configure ElasticSearch plugins with this path in `.magento/services.yaml`: `elasticsearch/configuration/plugins`.

RabbitMQ

You can either allow Commerce Cloud to provide ElasticSearch for you, or you can use the `QUEUE_CONFIGURATION` environment variable to specify an external queue.

4.2 DEMONSTRATE ABILITY TO USE SLAVE CONNECTIONS

How to leverage slave connections to MySQL, Redis

There are two environment variables that enable the use of a read-only connection. The default is `false`, but this speeds up responses from MySQL or Redis.

- `MYSQL_USE_SLAVE_CONNECTION`
- `REDIS_USE_SLAVE_CONNECTION`

This is configured in `.magento.env.yaml` like:

```
stage:
    deploy:
        MYSQL_USE_SLAVE_CONNECTION: true
```

For Pro accounts, Magento recommends that both Redis and MySQL slave connections are enabled.

Note that for Redis, the read-only connection is not available in integration. It is also **not** available if you specify custom details in the `CACHE_CONFIGURATION` environment variable.

Further Reading:

Deploy variables

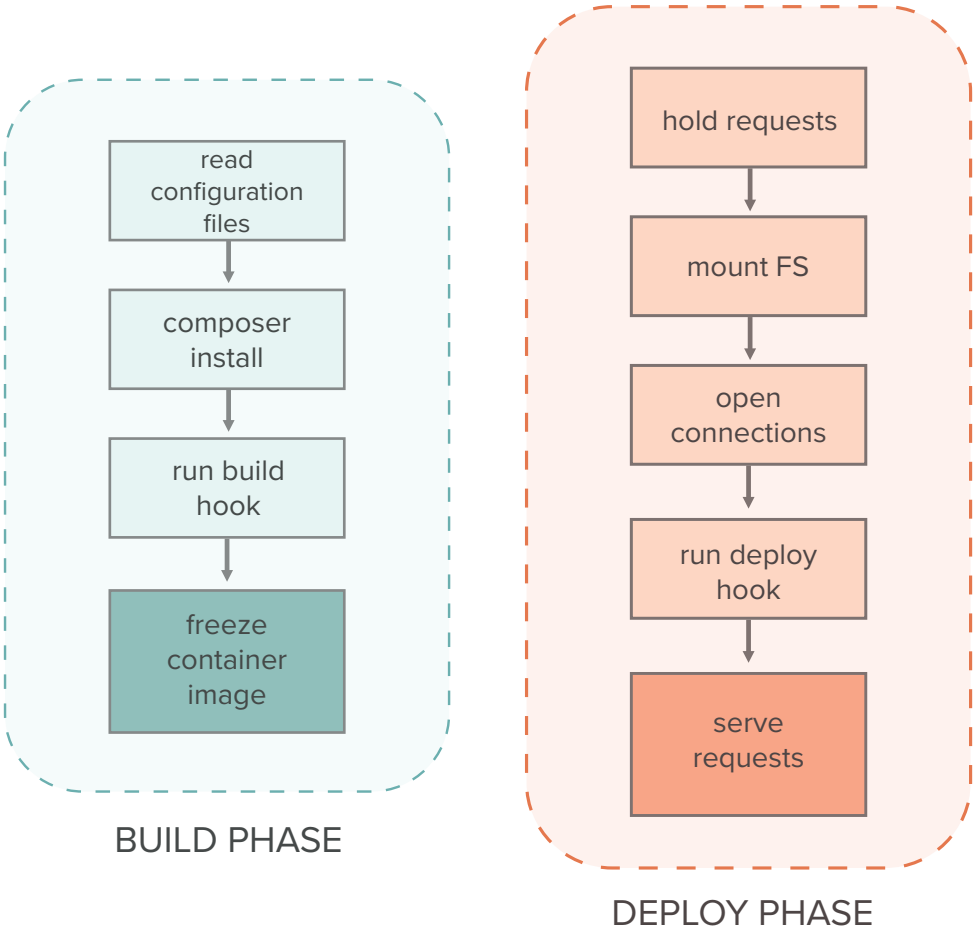
5 DEPLOYMENT PROCESS

10% of the test

SWIFT OTTER SOLUTIONS

5.1 DETERMINE THE PROCESSES DURING DEPLOYMENT

Describe all processes that are running during deployment: Build, deploy and post-deploy phases.



Build

The build process is configured in `.magento.app.yaml` with additional configurations specified in `.magento.env.yaml`. The primary command to execute build instructions is found in `.magento.app.yaml`, `hooks/build` and is:

```
hooks:
    build: |
        php ./vendor/bin/ece-tools build
```

Before this command is run, Magento's build system starts by running `composer install`. It is important to note that both the database and other services (like Redis/ElasticSearch, etc.) are not present during the build phase.

Overview of build execution:

- `composer install`
- Run commands found in `.magento.app.yaml`, `hooks/build`.
    - At some point, `php ./vendor/bin/ece-tools build` must be run. The `ece-tools build` section below goes into great detail on how this happens.
- Run any custom commands that you would like.

`ece-tools build`

This command is an alias to run two commands: `ece-tools build:generate` and `ece-tools build:transfer`.

`ece-tools build:generate`

One of my favorite ways to learn a system from the inside out (as is required to pass such a test) is to dive into the code wherever possible. The lack of actual code makes the Cloud Developer test more challenging because we must rely

on documentation. Some of that documentation is out of date, so we must then rely on experimentation.

However, the `ece-tools` package is an open-source PHP library included with composer so it is easy to open and understand. We will be doing a deep-dive into this project below.

This `build:generate` command executes a series of subcommands (the configuration is found in `vendor/magento/ece-tools/src/App/Container.php`).

DO NOT memorize the method names. Rather, understand the order in which they are executed and how they affect the system. After this section, we will look at some practical implications of this process.

- `PreBuild`:
  Outputs the notice "Verbosity level is ..."
  Cleans the `generated/code` directory.
  Cleans the `generated/metadata` directory.
- `SetProductionMode`:
  Uses the `DeployConfigWriter` to write to `env.php` (see here)
- `ValidateConfiguration`
  This executes a list of validators:
  - `ComposerFile`: this ensures that `Zend\Mvc\Controller\` is in `autoload/psr-4`. Failing this is a critical failure.
  - `StageConfig`: this ensures that the `stage` section in `.magento.env.yaml` is accurate (if it exists). Failing this is a critical failure.
  - `BuildOptionsIni`: validates the deprecated `build_options.ini` file. Failing this is a critical failure.

- `ConfigFileExists`: checks to see if `app/etc/config.php` exists. This file, as you might know, enables/disables modules, plus contains other important information such as basic website/store/store view details. If `app/etc/config.php` doesn't exist, the build will proceed, but you will get a warning.
- `DeprecatedBuildOptionsIni`: this file throws a warning if you are still using the old, defunct, out-of-date, and somewhat useless `build_options.ini` file.
- `StageConfigDeprecatedVariables`: this file lets you know if there are deprecated variables in `.magento.env.yaml`. This list of variables is found in `Schema::getDeprecatedSchema()`. At the time of writing, the only deprecated item is `SCD_EXCLUDE_THEMES`. You should instead use `SCD_MATRIX`.
- `ModulesExist`: this file ensures that `modules` is a key inside `app/etc/config.php`.
- `AppropriateVersion`: to use the `SCD_STRATEGY` and `SCD_MAX_EXECUTION_TIME`. See here and here.
- `ScdOptionsIgnorance`: this checks to see if `SCD_ON_DEMAND` or `SKIP_SCD` variables are configured. If they are, then a warning will display like:

  "When `SCD_ON_DEMAND` variable is enabled, static content deployment does not run during the build phase and the following variables are ignored: `SCD_STRATEGY`, `SCD_THREADS`, `SCD_EXCLUDE_THEMES`."

  Also, this will throw an exception if no websites/stores are configured in `app/etc/config.php`.
- `IdealState`: the Ideal State is when:

  1) both `SCD_ON_DEMAND` `SKIP_SCD` are not enabled,

  2) the `ece-tools post-deploy` hook is present in `.magento.app.yaml` (see here) and

3) `SKIP_HTML_MINIFICATION` is **not** enabled.

In this case, static content deploy will occur during the build phase which significantly reduces downtime when deploying to production (really a no-brainer for any type of in-production websites).

- `ModuleRefresh`: this enables all modules (see here).
- `ApplyPatches`: this applies patches and configures the file system. (see here):
  - `copyStaticFile`: this copies `pub/static.php` to `pub/front-static.php`.
  - `applyComposerPatches`: applies composer module patches found in the root directory's `patches.json` directory. Specify the composer module name and a version number. This applies patches found wherever you place them in your project (documentation recommends `patches/`). See this file for how they are applied.
  - `applyHotFixes`: this applies all `.patch` files found in the `m2-hotfixes` directory.
- `MarshallFiles`: this file:
  - Deletes the `var/cache` directory.
  - Copies `di.xml` files if < Magento 2.2.
- `CopySampleData`: if `vendor/magento/sample-data-media` is found, the media files are copied to `pub/media`.
- `CompileDi`: runs `bin/magento setup:di:compile`.
- `ComposerDumpAutoload`: this regenerates Composer's `autoload_classmap.php` file. This is necessary because additional files may have been included since Composer last generated the autoload classmap.

- `DeployStaticContent`: if `ece-tools` can, it will deploy static content during the build phase. This is desirable because this saves precious time during the deploy phase (i.e. when the site is down). For this to work, you must:
  - `SCD_ON_DEMAND` must **NOT** be enabled.
  - `SKIP_SCD` must **NOT** be enabled.
  - Websites and stores scope definitions **must be present** in `app/etc/config.php`. Remember to use the `php vendor/bin/m2-ece-scd-dump` command to place the appropriate and necessary scopes into `app/etc/config.php` (don't use `bin/magento app:config:dump` as this will dump all configuration into `app/etc/config.php` and `app/etc/env.php`).

    Then, this command runs `bin/magento setup:static-content:deploy` (see here).

Further Reading:

Review this to understand how the `.magento.env.yaml` is validated.

`ece-tools build:transfer`

Once the generation phase of the build completes, the next step is to prepare the "slug" (artifact, archive) for deployment.

- `CompressStaticContent`: this feeds the `SCD_COMPRESSION_LEVEL`, `SCD_COMPRESSION_TIMEOUT` and `VERBOSE_COMMANDS` variables into the `StaticContentCompressor`. Here is a sample compression command:

```
timeout -k 30 30 bash -c \
    find pub/static -type d -name DELETING_* -prune -o -type f
-size +300c \
     '(' -name '*.js' -or -name '*.css' -r -name '*.svg' \
     -or -name '*.html' -or -name '*.htm' ')' -print0 \
      | xargs -0 -n100 -P16 gzip -q --keep -4
```

- `ClearInitDirectory`: this clears the contents of the `init/` directory and removes `app/etc/env.php`.
- `BackupData`: this copies static content and writable directories to the 'init' directory:
  - `StaticContent`: if static content is deployed during the build process, the `pub/static` directory is temporarily moved to `init/pub/static` for the build directory.
  - `WritableDirectories`: these are not the mount points from `.magento.app.yaml` but rather this list (`getWritableDirectories` in `here`):

    `app/etc`

    `pub/media`

    `var/log`

    `var/view_preprocessed`

    These files are moved with the same directory structure into the `init` directory. Note that during deploy, the static content directories are symlinked into the `init` directory IF `SCD_ON_DEMAND` is not set.

Practical experience

- `composer install` runs before the execution of `hooks/build`.
- Patches from the `m2-hotfixes` directory are applied in the `ece-tools build` command.
- You can split up the `build` command into two commands: `build:generate` and `build:transfer` if you need to inject code between the execution of these commands.

Further Reading:

Phase 2: Build

Phase 3: Prepare the slug

Deploy

The deploy process has two parts: 1) it mounts the file system, configures connections and services, and then 2) it executes the deploy commands in `.magento.app.yaml`.

It is important to note that all incoming traffic is frozen for 60 seconds. This is so Cloud is properly re-configured to accommodate any routing changes introduced by the new environment.

To better understand what happens with the deploy hooks, let's take another deep dive into this area.

The `Deploy` class iterates through commands as specified in the `Container`.

- `PreDeploy`: this runs a series of commands:

- `ConfigUpdate\Cache`: this updates the cache configuration in `app/etc/env.php`. See this file.
- If the `CACHE_CONFIGURATION` environment variable is valid and cache details that are specified in `app/etc/env.php` are correct, no changes happen. One thing to note is that if you are using a Redis slave (`REDIS_USE_SLAVE_CONNECTION`), those details will not be updated in `app/etc/env.php`.

  If there is no Redis `relationship` (in `.magento.app.yaml`), nothing else happens.

  Otherwise, `app/etc/env.php` is updated with both the cache (`default`) details AND `page_cache` information.
- `CleanStaticContent`: if static content was built already or the environment variables list is missing the `CLEAN_STATIC_FILES` variable, then nothing happens. Please read up on cleaning static files as this can be helpful during development but cause interesting problems thereafter. If this proceeds... well, it does exactly what you think it will do: cleans the content from the `pub/static` directory.
- `CleanViewPreprocessed`: if the `SKIP_HTML_MINIFICATION` is absent (i.e. HTML is minified), this does nothing. Otherwise, the `var/view_preprocessed` directory is cleaned.
- `CleanRedisCache`: clears all caches (the `default` and `page_cache`).
- `CleanFileCache`: cleans the `var/cache` directory, if it exists.
- `RestoreWritableDirectories`: this uses the `RecoverableDirectoryList` to determine the list of files to copy or symlink and then uses the `BuildDirCopier` to copy/symlink these files/directories from the `init/` directory.

  The files in `init/app/etc/*` and `init/pub/media` are copied to their root-level locations.

  If static content was built in the build phase, then:

        - If the HTML is minified, copy the `init/var/view_preprocessed` directory.
        - Copy all files in `init/pub/static`.
        - It is important to note the copy strategy. If `CLEAN_STATIC_FILES` is set (by default it is), then the entire directory is copied. Otherwise, just the subfolders. This is helpful to retain existing files but isn't a good idea to leave on in production.
    - `SetProductionMode`: as you might guess, this enables production mode in `app/etc/env.php`.
- `DisableCron`: this turns off cron by updating `app/etc/env.php`.
- `ValidateConfiguration`: this goes through a list of configuration validations:
    - `DatabaseConfiguration`: determines if the `DATABASE_CONFIGURATION` property is set. If this is set to `_merge`, then the `host`, `dbname`, `username` and `password` must all be set in the `DATABASE_CONFIGURATION`. The Cloud environment configures it, but you can add extra properties in `.magento.app.yaml`'s `stage/deploy/DATABASE_CONFIGURATION` section. Failure results in a failed deploy.
    - `SearchConfiguration`: this merges/configures the `SEARCH_CONFIGURATION` parameters in the `.magento.app.yaml` file. Failure results in a failed deploy.
    - `ResourceConfiguration`: this maps connection names in `app/etc/env.php` with database connections. See `RESOURCE_CONFIGURATION`. Failure results in a failed deploy.
    - `SessionConfiguration`: this configures Redis for utilization in session storage. See `SESSION_CONFIGURATION`. Failure results in a failed deploy.
    - `ElasticSuiteIntegrity`: if ElasticSearch is enabled, ensure that it is properly installed and the `SEARCH_CONFIGURATION` is set to `elasticsearch`. Failure results in a failed deploy.

- `AdminData`: the admin data environment variables (`ADMIN_FIRSTNAME`, `ADMIN_LASTNAME`, `ADMIN_EMAIL`, `ADMIN_USERNAME`, `ADMIN_PASSWORD`) are used when installing Magento. These variables are also used to create a new Magento admin user (`ADMIN_EMAIL` is then required).
- `PhpVersion`: ensures that the environment's PHP version is compatible with the currently-installed version of `magento/magento2-base`. Note that it is very important to run `composer update` in an environment that has the same version of PHP as is specified in `.magento.app.yaml`'s `type`.
- `SolrIntegrity`: ensures that Solr is NOT configured. Note that this does not stop a deploy but rather adds a message to the log files.
- `ElasticSearchUsage`: if ElasticSearch is installed, but not used, this throws a warning to eliminate it from the tech stack.
- `ElasticSearchVersion`: checks to see if the correct version of ElasticSearch is installed.
- `AppropriateVersion`: if Magento's version is less than 2.2, warnings are shown for several variables (`CRON_CONSUMERS_RUNNER`, `SCD_STRATEGY` and `SCD_MAX_EXEC_TIME`). Also, if Magento's version is NOT 2.1 (so 2.2 or better), a warning is shown that the `GENERATED_CODE_SYMLINK` is not available for `this version`. The `generated/` directory is no longer writable and thus you cannot run `setup:di:compile` on the server.
- `ScdOptionsIgnorance`: checks to see if static content should be built on deploy (not a good idea). This is determined in `ScdOnDeploy`. If any of the following are set, static content will not be built on deploy: 1) `SCD_ON_DEMAND` in `.magento.env.yaml` is set, 2) `SKIP_SCD` in `.magento.env.yaml:stage/deploy` is set or 3) static content was built during the build phase.

  This validator returns an error when static content can't be built (see above) and the `SCD_STRATEGY`, `SCD_THREADS` or the deprecated `SCD_EXCLUDE_THEMES` are set. This is simply a notification that these variables are ignored.

    - `DeprecatedVariables`: sends a warning to the logs if these deprecated variables are used:

      ```
      VERBOSE_COMMANDS === enabled
      SCD_EXCLUDE_THEMES
      ```

      Note that there are a couple of other variables, but there is no documentation on them.

    - `RawEnvVariable`: if the `SCD_THREADS` variable has a non-numeric value, this throws a warning.

    - `MagentoCloudVariables`: validates whether or not environment/deploy variables are set to the correct value type.

    - `JsonFormatVariable`: ensures that non-array variables can be decoded as json, if necessary.

- `UnlockCronJobs`: if Magento 2.2.2 or greater is installed, this happens automatically. Essentially a setup upgrade command can cause cron jobs to lock up. This sets all running cron jobs to the error status so the upgrade can `proceed`.

- `SetCryptKey`: if an encryption key is NOT specified in `app/etc/env.php` and one is specified in the `CRYPT_KEY` environment variable, it is saved to `app/etc/env.php`. Note that this should only happen on the installation of a Magento site. Otherwise, API keys and any other configuration that is encrypted > decrypted will be rendered useless. Your website's UPS rate calculator will no longer return rates.

- `InstallUpdate`: if Magento is not installed (tables don't exist or the `install/date` node is missing in `app/etc/env.php`), install Magento:

    - `Setup`: this runs `setup:install`. Please familiarize yourself with the defaults. Note that for an automated install, you must have specified the `ADMIN_EMAIL` environment variable. Also, by default, sessions are

saved to the database. The connection information is loaded from the configuration that is then merged with environment variables.

- `ConfigUpdate` runs:

  `PrepareConfig`: this updates 'app/etc/env.php' with the values from `SCD_ON_DEMAND` (`static_content_on_demand_in_production` is written), `SKIP_HTML_MINIFICATION` (`force_html_minification` is written) and `X_FRAME_CONFIGURATION` (`x-frame-options` is written).

  `CronConsumersRunner`: this updates the `cron_consumers_runner` configuration in `app/etc/env.php` (see here).

  `DbConnection`: if a database host is configured (i.e. not in the build environment still), slaves are utilized, and database connection information is specified in `app/etc/env.php` (or the `_merge` flag is specified), `app/etc/env.php` is updated with the slave database-connection information.

  `Amqp`: this merges in RabbitMQ details.

  `Session`: this configures session storage information if Redis is enabled. You can use the `SESSION_CONFIGURATION` deploy variable to adjust the configuration.

  `SearchEngine`: updates the `system/default/catalog/search/engine` (see here) Store Configuration value to the configured value in `SEARCH_CONFIGURATION`. Note that once this value is set in `app/etc/env.php` it can no longer be changed in the admin.

`Urls`: if this is NOT a master environment (see `isMasterBranch` in here) and the `UPDATE_URLS` variable is enabled (which it is by default), the URL values in `core_config_data` and `app/etc/env.php` are updated with the URLs as found in the `UrlManager`. These values come from the `ENV_ROUTES` environment variable.

`DocumentRoot`: configures the `directories/document_root_is_pub` to be true in `app/etc/env.php`.

`Lock`: this sets the mount point for the lock provider from the `MAGENTO_CLOUD_LOCKS_DIR`. If no directory is specified, then Magento uses the database. The configuration is concocted in here.

- `ConfigImport`: this runs `bin/magento app:config:import` to import configuration details from `app/etc/env.php` and `app/etc/config.php` into the database. See here.
- `ResetPassword`: unlike this class's name, this file does not reset the admin password. Instead, this notifies the admin of the admin URL. This only runs at install time.

If Magento IS installed, an update operation must now take place. The parent class is this one. However, this simply loops through the processes outlined in `App/Container.php`.

- `ConfigUpdate`: see above in the install processes.
- `SetAdminUrl`: if the admin url is defined as an environment variable, the `backend/frontName` will be configured with this value.
- `Setup`: this runs the ubiquitous `bin/magento setup:upgrade --keep-generated`.

- `DeployStaticContent`: building static content in the deploy phase is the

default place for this action. With some minimal configuration, you can move it to the ideal time of the build phase.

Before we dive into the details of this, it's important to understand how some of the configuration variables affect whether or not this runs. First, if you use `SCD_ON_DEMAND`, the static content is cleaned. This is the `pub/static` and `var/view_preprocessed` directories. If the `SKIP_SCD` configuration is set or the static content deploy happened in the build phase, this is skipped. If you are using the `CLEAN_STATIC_FILES` flag, then, similar to above, the static content is cleared out. Otherwise, here is what happens:

`Generate`: first, this checks if the `pub/static` directory is writable. It might not be writable if you have removed the `mounts/pub/static` entry.

This then calls `CommandFactory`. This file generates the `bin/magento setup:static-content:deploy` commands. The first command runs the deploy for all themes EXCLUDING those listed in the matrix (more details here). Subsequent commands are run for each theme in the matrix.

It is important to note that by default, `setup:static-content:deploy` generates all content for all themes for all locales. So, the first deploy generates everything EXCEPT for themes specified in the matrix. Then, each theme in the matrix is executed for each language specified.

The goal of this is to speed up the time to generate the build files. Otherwise, this can take quite a while.

It is important to note that this is where the `SCD_THREADS` and `SCD_STRATEGY` are utilized. The `Option` class also takes the `SCD_MAX_EXEC_TIME` and applies it to the `--max-execution-time` class.

- `CompressStaticContent`: IF `SCD_ON_DEMAND` is selected, no compression will happen. Also, IF `SKIP_SCD` or static content was already built in the build phase, no compression will happen. Otherwise, compression will be executed on the contents of the pub/static directory (see this command in the build phase for more details).

- `DisableGoogleAnalytics`: if the staging or integration branches and `ENABLE_GOOGLE_ANALYTICS` are disabled (default), then Google Analytics is disabled.

- `DeployCompletion`: this only runs if there is no post-deploy hook (i.e. the `post_deploy` node configured in `hooks` in `.magento.app.yaml`). This command:
  `EnableCron`: this removes the `cron/enabled` flag from `app/etc/env.php`.
  `Backup`: this backs up `app/etc/env.php` and `app/etc/config.php` by copying them and appending the `.bak` extension:

```
drwxr-xr-x 3 web web 4.0K Jun 15 21:41 .
drwxr-xr-x 5 web web  131 May 27 19:11 ..
-rw-r--r-- 1 web web  863 Jun 25 13:04 NonComposerComponentRegistration
-rw-r--r-- 1 web web  14K Jun 25 13:05 config.php
-rw-r--r-- 1 web web  14K May 27 20:45 config.php.bak
-rw-r--r-- 1 web web  739 Jun 25 13:04 db_schema.xml
-rw-r--r-- 1 web web 119K Jun 25 13:04 di.xml
drwxr-xr-x 2 web web 4.0K Nov 16  2018 enterprise
-rw-r--r-- 1 web web 2.6K Jun 25 13:05 env.php
-rw-r--r-- 1 web web 3.0K May 27 20:45 env.php.bak
-rw-r--r-- 1 web web  516 Jun 25 13:04 registration_globlist.php
```

`CleanCache`: this runs `bin/magento cache:flush`.

Further Reading:

Phase 4: Deploy slugs and cluster

Phase 5: Deployment hooks

Post-Deploy

The final command that `ece-tools` provides for the build/deploy process is the `ece-tools post-deploy` command:

```
# .magento.app.yaml
hooks:
    # …
    post_deploy: |
        php ./vendor/bin/ece-tools post-deploy
```

The `PostDeploy` class is a shell to iterate through the multiple versions:

- `DebugLogging` validation: if this is the master branch and debug logging is enabled, a message is logged encouraging you to turn off debug logging.
- `EnableCron`: this does what you think it does: it enables cron.
- `Backup`: this copies `app/etc/env.php` and `app/etc/config.php`. The new files have a `.bak` appended to them.
- `CleanCache`: runs `bin/magento cache:flush`.
- `WarmUp`: this asynchronously crawls pages on the website. These URLs come

from `stage/post-deploy/WARM_UP_PAGES` configuration in `.magento.env.yaml` file. There are several types of patterns that can be utilized. Please review the documentation to understand how this works.

One thing to note is that there is an important variable that is not in `ece-tools`: `TTFB_TESTED_PAGES`. In the post-deploy process, Magento will test these specific pages and log the time to the first byte.

Explain why downtime occurs on your project

Downtime is a result of the deploy process taking longer than it should. The site goes into maintenance mode when the deploy phase begins (as well as all incoming requests being halted).

That said, incoming traffic is always frozen for 60 seconds.

Further Reading:

Cloud deployment process

Phase 4: Deploy slugs and cluster

What role every process/phase plays and how to impact every process

There are five phases to get code to be usable:

1. Validation: the Magento git server checks for valid configuration. Note that this does not happen immediately if using another git server, like GitLab or GitHub.

2. Build: see above for a very detailed description of what happens. Ideally, the static content deploy will happen in this phase. This reduces downtime when moving to production. No services, such as the database or Redis, are available during the build phase.

3. Prepare the slug: the filesystem for the Magento website is prepared. Remember, it is read-only, except for the mounts that are specific in `.magento.app.yaml`. Also, if static content is published in the build phase, this will be present in `init/pub/static`. Note that the `STATIC_CONTENT_SYMLINK`, which is `true` by default, creates symlinks for each directory INSIDE `pub/static`.

4. Deploy slugs and cluster: this is the part of the process where the network is "down" and the environment is about to be configured with the new code.

5. Deployment hooks: there are two parts to this process: 1) running the `deploy` hooks and then the 2) `post_deploy` hooks (when `post_deploy` hooks are run, the site is back up; however, it could still be a while before pages are loaded if the `SCD_ON_DEMAND` variable is set).

Further Reading:

Five phases of Integration build and deployment

Continuous integration

How Magento Cloud deploys Magento. What every script does on every deployment phase

See above.

How to extend these scripts and best practices for doing so

There is no built-in way to modify the functionality of the `ece-tools` scripts. However, a workaround (might this be the way that was intended?) is to use the `psr-4` section in `composer.json`. This would work very well for updating a single class.

You can also create a git patch and place it into the `m2-hotfixes` directory.

In addition, if you wish to add functionality to the process, you can simply create new commands and add them to the list in `hooks/[build|deploy|post_deploy]`. Note that in the build phase, the database is not available. Also, before the `build` hooks are run, `composer install` is run.

Describe the ways to retrieve logs for phases and its scripts

The deploy log is found under `/var/log/deploy.log`. Build and post-deploy logs are found in the message list for each environment.

You can also review `/app/var/log/cloud.log` as this provides logs for all phases in one file.

5.2 DEMONSTRATE THE ABILITY TO CREATE MAGENTO CLOUD SCRIPT CONFIGURATIONS

Which configurations you can set in .magento.env.yaml, and Cloud environment configurations that are not described in the units about SCD and service configurations

- ADMIN variables: these are configured in the Project Web Interface. These are used to configure admin users when Magento is being installed. After Magento is installed, you can use `bin/magento admin:user:create` (while this is named `create`, if you specify a username of an existing Magento user, an update will happen instead).
- All other variables are configured in `.magento.env.yaml`.
- Setting a variable in the Project Web Interface automatically redeploys the environment.
- Important: Environment variables can be used to pre-set Magento configuration (see here and here). Environment variables are variables specified in the Project Web Interface that are prefixed with `env:`.

Keep in mind that `.magento.env.yaml` is updated every time the `ece-tools` Composer package is updated. Here is a list of all variables that have not been discussed yet in this chapter:

stage/global:

- `MIN_LOGGING_LEVEL`: used to gain more insight into errors (increases the verbosity of logging). Set this to be `debug` when necessary.
- `X_FRAME_CONFIGURATION`: this updates the `x-frame-options` setting in `app/etc/env.php`.

stage/build:

- `SCD_COMPRESSION_LEVEL`: how compressed static content should be. Default is 6. See this discussion in the build scripts.
- `SCD_COMPRESSION_TIMEOUT`: the maximum amount of time for compressing files.
- `SCD_EXCLUDE_THEMES`: deprecated (i.e. don't use this). This was before the matrix.
- `SCD_MATRIX`: specifies which themes and languages to build.
- `VERBOSE_COMMANDS`: this specifies the verbosity flag for commands.

stage/deploy:

Note: in-service configuration (like `DATABASE_CONFIGURATION` and `QUEUE_CONFIGURATION`) variables, you will see the `_merge` option. When set to `true`, the existing configuration in `app/etc/env.php` is merged with the configuration specified in this file.

- `CACHE_CONFIGURATION`: this is used to specify a cache provider (if not Redis) and the Redis databases required.
- `CLEAN_STATIC_FILES`: clears the `pub/static` directory.
- `CRON_CONSUMERS_RUNNER`: ensures that message queues are running after deploy.
- `CRYPT_KEY`: if NO `crypt/key` is set in `app/etc/env.php` this allows you to specify one. **Note that this is specified in the Project Web Interface (not in `.magento.env.yaml`)**.

- `ENABLE_GOOGLE_ANALYTICS`: `true` enables Google Analytics in integration and staging.
- `FORCE_UPDATE_URLS`: this resets the Magento URLs during deployment **to production or staging (Starter) or to any Pro environment** (default is true).
- `SKIP_SCD`: this prevents static content deployment from happening during the deploy phase. This is a good idea as it reduces downtime when deploying code to production.
- `STATIC_CONTENT_SYMLINK`: by default, static content is symlinked to `init/pub/static`. This allows the three-server configuration on Pro plans to utilize the same source for static content. If you enable on-demand static content deployment (`SCD_ON_DEMAND`), this is disabled. Why? Because on-demand (in production) deployment of static content involves writing files. The `init/` directory is not writable and thus files must be written directly into `pub/static`.
- `UPDATE_URLS`: this updates URLs locally and in integration on Starter plans.
- `VERBOSE_COMMANDS`: This sends verbose commands to the logs. There is also a helpful `log` section in `.magento.env.yaml` where you can configure logging details (including email and Slack alerts).

Further Reading:

Ece tools

Working with variables

6 STATIC CONTENT DEPLOYMENT

10% of the test

SWIFT OTTER SOLUTIONS

6.1 DEMONSTRATE ABILITY TO MOVE SCD TO BUILD PHASE

Describe the default process of generating SCD and how it impacts downtime

We already covered this in great detail in Objective 5. However, for the sake of review, we will cover it again in an overview format.

Both the `build` and `deploy` phases have the capability to generate static content. To generate static content, Magento must have a knowledge of what themes and locales are in use. This requires a connection to the database UNLESS you have used `./vendor/bin/ece-tools config:dump`. Because the database is readily available in the deploy phase, this is the easiest time to generate the static content.

As we all learn in life, the easiest is not always the best (rarely it is). The nice thing is that it isn't that hard to build static content in the build pipeline.

The process to enable static content deploy in the build phase is:

- SSH into production
- Run `./vendor/bin/ece-tools config:dump`
- Copy `app/etc/config.php` into local project
- Commit
- Deploy

Smart Wizards

Smart wizards are a way to optimize the deployment or operation of a Cloud-hosted website.

There are several available:

- `./vendor/bin/ece-tools wizard:ideal-state` this ensures that the static-content deploy happens in the build stage, the post-deploy hook is configured and `SKIP_HTML_MINIFICATION` is true. Note that this check is run as a validator in the `build:generate` command (if it fails, a warning is output and it does not stop the build process).
- `./vendor/bin/ece-tools wizard:master-slave`: this checks to see if MySQL and Redis slave connections are configured.
- `./vendor/bin/ece-tools wizard:scd-on-demand`: writes "SCD on demand is (enabled or disabled)."
- `./vendor/bin/ece-tools wizard:scd-on-build`: utilizes the global stage `ScdOnBuild` validator to see if: `SCD_ON_DEMAND` is disabled, `SKIP_SCD` is disabled and that `app/etc/config.php` exists and contains scope configurations.
- `./vendor/bin/ece-tools wizard:scd-on-deploy`: uses the global stage `ScdOnDeploy` validator to see if: `SCD_ON_DEMAND` is disabled, `SKIP_SCD` is disabled and build-phase SCD is disabled.

Describe the reason for moving generation static content to the build phase. Consider the way to do this and show the result (timeline)

Reduce downtime. You could also argue that pre-generated static content is more secure because you cannot change the static content files. This reduces the attack vector of MageCart-esque problems where files are updated to inject malicious code.

The `./vendor/bin/ece-tools config:dump` command runs `bin/magento app:config:dump`, then filters configuration to ensure that only stores and websites are saved `bin/magento app:config:import`. Because `app/etc/config.php` should be included in the git repository, refresh your local copy of `app/etc/config.php` with the latest generated in production.

Determine additional configuration that helps decrease deployment time (`SKIP_HTML_MINIFICATION`)

Having `SKIP_HTML_MINIFICATION` set to `true` (which is the default) prevents an additional file copy in the deploy phase. If HTML is minified, files are copied from `init/var/view_preprocessed` into `var/view_preprocessed`. This takes some (minimal) time. As such, HTML is not minified by default. It is up to the merchant if the fewer downloaded bytes is worth the slightly longer deploy.

Unfortunately, after dealing with static content deploy, the `setup:upgrade` command must be run, and I do not see a way around this.

Further Reading:

Zero downtime deployment

Update Database (this demonstrates a command that you can run to avoid the downtime that `setup:upgrade` generates when running this command is not required).

Display time measurements

This table (thank you, Billy) represents static content deploy times for a vanilla Magento install on an integration branch with sample data.

	SCD on deploy, skip minify	SCD on deploy, minify	SCD on build, skip minify	SCD on build, minify	SCD on Demand
Build	01:11	01:10	01:52	01:58	01:14
Deploy	06:46	07:31	00:48	01:46	00:39
Post deploy	00:02	00:02	00:02	00:02	00:02
Total	08:37	09:20	03:22	04:27	02:31
SCD time:	05:18	06:04	00:41	00:41	00:00

For a basic Magento Cloud deploy, building the static content on deploy takes almost a minute extra (i.e. downtime). Total time for this deploy was 107 seconds.

When building the static content in the build phase, the total was 44 seconds (most of it is the `bin/magento setup:upgrade` command). Keep in mind that the disparity will be much greater (static content deploy in deploy phase) if there are multiple themes and locales.

Practical experience:

- Ensure there is no scope (stores/websites/groups) data in `app/etc/config.php`. Deploy code to Cloud. Check the time between when maintenance mode is enabled and when it is then disabled.
- Then, use the `ece-tools config:dump` to generate a relevant `app/`

`etc/config.php` file. Move this down to your repository and push it up. Static content should now be built in the build phase. If you are working on a non-vanilla Cloud project, ensure that `SCD_ON_DEMAND === false`, `SKIP_HTML_MINIFICATION === true`, `SKIP_SCD === false` (in the `.magento.env.yaml` build stage) and `SCD_STRATEGY === compact`.

6.2 DEMONSTRATE ABILITY TO AVOID SCD ON BOTH PHASES

What are cases for avoiding SCD

Avoiding static content deploy does speed up build and deploy processes (the static files from the previous build will still be present). It gets code faster to Cloud during the development phase of a project.

Describe the way to do this in all cases, and the expected result

You can specify the `SKIP_SCD` environment variable in `.magento.env.yaml` (both the `build` and `deploy` nodes).

Note that the contents of the `pub/static` directory will be preserved from the last run. If you previously were deploying static content in the build phase, your `pub/static` directory will contain three broken symlinks (not ideal). Manually running `bin/magento setup:static-content:deploy` does not work in production.

The best approach is to either utilize `SCD_ON_DEMAND` or use `SKIP_SCD` on just the `build` phase. Once this content is in the environment and no symlinks are present, you can enable `SKIP_SCD` on the `deploy` phase.

6.3 DESCRIBE HOW TO GENERATE STATIC CONTENT ON DEMAND

Describe the default Magento behavior in Production mode and why the new "mode" was added

By default, production mode disables generating any static content files. This speeds uploading content from the website as Magento doesn't have to look for new content.

This would prevent on-demand deploy of this static content. However, as with everything "Magento," there's a way around this.

Further Reading:

Setting the SCD on build

How does Magento behave when it is set to this "mode"?

When deploying in on-demand mode, the deploy scripts add `static_content_on_demand_in_production` to `app/etc/env.php` (see Objective 5).

Even more important is that patches are applied to Magento so that it understands this flag. This is somewhat present in the core. Here are the patches that are applied:

- `configure_scd_on_demand`: if in production mode and `static_content_on_demand_in_production` is configured, then the asset can be created. It also allows for these newly generated files to be minified.
- `respect_minification_override`: adds `or` logic so if production SCD is enabled OR `force_html_minification` is enabled, the file is minified.
- `unlock_locale_editing`: when in production mode, the Locale is locked (in Store > Configuration > General > Locale Options) to the locales for which themes have been generated. This unlocks it so that you can generate static content for any locale.
- `trim_static_content`: this removes `/static/version[NUMBER HERE]` from the `$_GET['resource']`.

Know when users can use this configuration and how it works from the Cloud side

This should only be done in the development (pre-release) phase of a project. Ultimately, inconsistencies could arise as there is no guarantee that these files will be identical to what is intended. While the risk is small, there is some possibility of attack as the static files are no longer read-only.

Display time measurements

Generating the static content files takes time. For example, `styles-m.css` took 9.4s. `styles-l.css` took 4.54s. Once they are generated, they take < 300ms.

Practical experience:

- Deploy your development project with `SCD_ON_DEMAND` enabled.
- Switch the locale for the frontend.

7 DEVELOPMENT

20% of the test

SWIFT OTTER SOLUTIONS

7.1 DEMONSTRATE ABILITY TO CHANGE CONFIGURATIONS

What are the sources of Magento configuration, and which priorities have different sources of Magento configuration?

- Environment variables (these override the previous configuration sources). For example, to change the `carriers/tablerate/active` for all stores, use the environment variable, `CONFIG__DEFAULT__CARRIERS__TABLERATE__ACTIVE`. You can also use it for a single store like `CONFIG__STORES__DEFAULT__CARRIERS__TABLERATE__ACTIVE` (use the code for the store in place of `\DEFAULT`). You can specify this per-environment in the Project Web Interface with `env:`. Any time you specify a new variable for an environment, that environment is redeployed. You can also allow child environments to inherit this value, which will save you having to replicate this across environments.
- `app/etc/config.php`: this file contains information that is used across all environments (this is specified in the `system/default` path; use `system/stores/[store code]` for an individual store). Any configuration that is specified here is merged with `app/etc/env.php`.
- `app/etc/env.php`: this file is specific to each environment. Side note: don't specify environment variables for encrypted store configuration values. While it works, websites should have different encryption keys, and this can create problems where Magento can't legibly decrypt the value (i.e. different encryption key).
- Database
- Module XML configuration

Magento configuration has an order of operations. Many values are pre-configured in module XML files (`etc/config.xml`), which can be overridden by area-specific XML files. A Magento admin can go to Stores > Configuration and

set a new value for any configuration values that are exposed in the module's `etc/adminhtml/system.xml` file.

You can override these values in the `app/etc/env.php` file or `app/etc/config.php` files or with environment variables. When a Store Configuration value is overridden in one of these three areas, the Magento admin is unable to change the value. For example, running `bin/magento app:config:dump` effectively locks all Magento configuration because all values are now specified in `app/etc/config.php`. One thing to remember is that `app/etc/env.php` and `app/etc/config.php` are merged together at runtime.

Further Reading:

Use environment variables to override configuration settings

Configuration management for store settings

Magento's deployment configuration

7.2 DEMONSTRATE ABILITY TO CHANGE A LOCALE

Know how to change a locale on Cloud

The challenge with changing locales is that Magento locks the locales to ones that have their static content generated when in production mode.

If you wish to change locales, the key is to enable `SCD_ON_DEMAND` in the integration environment. You can do that in the Project Web Interface by going to Settings > Variables > Add `SCD_ON_DEMAND = true`. This will redeploy the

store and allow you to change the locale. Then, dump the configuration (`./vendor/bin/ece-tools config:dump`) and integrate the updated `app/etc/config.php` into your git repository.

Further Reading:

Configuration management for store settings

7.3 DEMONSTRATE ABILITY TO ADD EXTENSIONS

Know how to install Magento extensions and themes (limitations, read-only filesystem, etc.)

To install a new extension or theme, simply check out a new branch from `master` (Starter) or `integration` (Pro). Either drop the files in `app/code` or use Composer to install (preferred). Note that with Composer, you can install from a git repository (like GitLab), using the VCS repository configuration.

Keep in mind that it is imperative to run `composer install` or `composer update` in an environment that matches the PHP version of the deployed site. That is because the composer dependencies are established in `composer.lock` and then `composer install` (what is run before the build hooks) installs, or at least tries to install those dependencies.

Modules must be built correctly (Marketplace extensions are usually good). For example, if a module uses the `ObjectManager` outside of the constructor, there is a reasonable chance that files will be written to the `generated/`

directory. That would normally work except the `generated/` is read-only. Trying to write to the `generated/` directory when it is read-only will result in a fatal error.

You need to enable the module. Because `app/etc/config.php` (where the modules are enabled) should be saved in git, you simply need to run `bin/magento module:enable [MODULE_NAME]` and then commit and push.

Side note: I am confident that you know not to use `ObjectManager` in your code (except in the 0.01% of use cases). It's a bad idea. And this is a demonstration of why it is bad. Please save yourself the embarrassment of having someone else come in to find you added code smells to your code (or merchants are upset asking why their site breaks when going to production).

Installing a theme is basically identical, except that a manually-built theme resides in `app/design/[AREA]`.

Further Reading:

- Install, manage, and upgrade extensions
- Install a theme
- Starter develop and deploy workflow
- Pro develop and deploy workflow

7.4 DEMONSTRATE ABILITY TO ENABLE / DISABLE A MODULE

Know how to enable or disable a module on Cloud

Use `bin/magento module:disable` locally, commit and push.

Remember that the `app/etc/` directory is writable.

7.5 DEMONSTRATE ABILITY TO SET UP A MULTISITE CONFIGURATION

Know how to setup multisite configuration: Adding and configuring new websites in Magento; Nginx configuration through the `.magento.app.yaml` for multisite setup; how to route websites through the magentovars.php

Routes

If you intend to set up Magento stores with subdomains, you need to route these back into the main Cloud backend. This happens in `.magento/routes.yaml`. You can use a wildcard (`*`) or hardcode in the subdomains:

```
# .magento/routes.yaml

"http://b2b.{default}/":
    type: upstream
    upstream: "mymagento:php"
```

Note that `mymagento` is specified in `.magento.app.yaml` in the `name` node.

`.magento.app.yaml`

To configure Magento to serve different sites as a subfolder, you need to update the `web/locations` configuration in `.magento.app.yaml`.

`magento-vars.php`

This is a powerful file that is run before `pub/index.php`. As such, you can configure most (all?) server, post, and get variable going into Magento. While use cases are limited, they do save you having to modify `pub/index.php` (and maintaining those modifications through composer updates).

This file has access to everything in the `$_SERVER` superglobal. This includes environment variables. Here are a few (for the url http://staging-55555-vvvvvvv.us-4.magentosite.cloud/b2b/):

- `MAGENTO_CLOUD_APPLICATION`: the base64-encoded version of the JSON representation of `.magento.app.yaml`.
- `MAGENTO_CLOUD_ROUTES`: the base-64-encoded version of all routes available to the Magento application.

- `HTTP_HOST`: for example `staging-55555-vvvvvvv.us-4.magentosite.cloud`
- `HTTP_X_ORIGINAL_ROUTE`: `http://{default}/`
- `DOCUMENT_URI`: `/b2b/index.php`
- `DOCUMENT_ROOT`: `app/pub/`

Here is an example of `magento-vars.php` to demonstrate such routing:

```
// magento-vars.php


if (isset($_SERVER['DOCUMENT_URI'] && stripos($_SERVER['DOCUMENT_URI'], '/b2b/')
!== false) {
    $_SERVER["MAGE_RUN_CODE"] = "b2b";
    $_SERVER["MAGE_RUN_TYPE"] = "website";
}
```

Practical experience:

- Configure multiple stores (don't forget updating `app/etc/config.php` in production and then syncing data back to staging and integration).
- Use `magento-vars.php` to `print_r($_SERVER)` to understand the variables that are available.
- Configure a `/b2b` URL path.

Further Reading:

Set up multiple websites or stores

7.6 DEMONSTRATE ABILITY TO USE VARIABLES

When do you need to use variables; which configurations you can change using variables

Variables are another way to control the website and the build process. There are a number of ways to configure variables (each has their own context).

For example, the `.magento.env.yaml` configures variables for different phases (`build`, `deploy`, `post-deploy`, and `global`). These variables configure the website or change how the process behaves. They have little use long-term.

Further Reading:

Environment variables

What is the difference between variables and environment variables

Variables are specified in `.magento.env.yaml`. Environment variables are configured per environment in the Project Web CLI and cascade to child environments (for example integration to all child integration environments).

Remember, to use an environment variable to change Magento configuration settings, the variable name must start with `env:`.

Further Reading:

Set environment and project variables

What is the difference between project and environment level variables

Project variables are set using `magento-cloud project:variable:set` and apply to all environments (including build). Once you create a project variable, you need to re-deploy each environment. This can be done with a blank commit: `git commit –allow-empty -m "Redeploying environments"`.

Environment variables are configured per environment and thus are usually unique per environment (they can contain sensitive data). They can cascade through child environments. In the Project Web Interface, you must select "Inheritable by child environments" to make this choice available.

Further Reading:

Working with variables

Magento Cloud CLI reference

8 TROUBLESHOOTING

10% of the test

SWIFT OTTER SOLUTIONS

8.1 DEMONSTRATE ABILITY TO LOCATE AND USE LOGS

Locate the Magento application logs. Locate system services logs on integration and Starter environments. Locate system services logs on Pro environments

Log file overview:

- `/var/log/access.log`: this stores, in JSON format, all visits to the website and assets.
- `/var/log/app.log`: **application service-related events (related to PHP-FPM)**. For example, if New Relic is not properly configured, those errors will be displayed here (because PHP-FPM pushed events up to New Relic).
- `/var/log/cron.log`: logs the `cron:run` execution and its results.
- `/var/log/deploy.log`: the results from the deploy process. Note that this does not display in the Project Web Interface because Cloud could be deploying to multiple environments (Pro). Note that this log file will be in a `platform` directory on Pro projects.
- `/var/log/error.log`: this contains web server errors. For Pro projects, you need to check this file on all three deploy nodes.
- `/var/log/php.access.log`: this contains PHP-FPM renders (their time, size, response code, etc.).
- `/var/log/post-deploy.log`: logs for the post-deploy hook. Note that these are viewable in the Project Web Interface.
- `/app/var/log`: typical Magento log files.

Other than the log file names (it's a good idea to understand what is in each log file), the big difference between Starter and Pro is that the service logs (like `app.log`) are stored in `/var/log/platform/.../app.log`.

Further Reading:

View logs for troubleshooting

Project structure

Pro Project log locations

Starter Project log locations

8.2 DEMONSTRATE ABILITY TO CREATE SNAPSHOTS AND BACKUPS

Snapshots are available for all Starter and Pro projects. They can be created using the Project Web Interface with the "snapshots" button (upper right-hand corner) or with the Magento Cloud CLI using the `magento-cloud snapshot:create` command.

You can also create a database dump using the `./vendor/bin/ece-tools db-dump` command. This uses the `DB\Dump` class and utilizes the `mysqldump` command.

8.3 DEMONSTRATE ABILITY TO DEBUG

How to use XDebug on Cloud

This requires a good deal of configuration, but it is quite worth it if you have major problems to troubleshoot. It goes without saying that you shouldn't use this in production. Instead, sync data and code from the `master` branch to a new integration branch and debug there.

First, you must include `xdebug` in the PHP extensions and deploy. Remember that these changes affect everything but Pro staging and production environments.

Second, you must configure PHPStorm. Path mapping basically is mapping the root project directory to the `/app` directory. For Pro projects, the directory is `/app/{PROJECT}` (append `_stg` for the staging environment).

Third, you must establish the SSH tunnel. I did it like this:

```
ssh -R 9000:localhost:9090 ssh-path--mymagento@ssh.us-4.magento.cloud
```

Note that the first port number is the port number on the server. The second one is the local port number (as configured in PHPStorm).

Fourth, install the Xdebug helper (and set the correct IDE key). Enable Listening for PHP Connections and start at first line.

Finally, enable Xdebug in the browser and refresh the page.

Practical experience:

- Configure and execute a Xdebug session.

Further Reading:

Configure Xdebug

8.4 DEMONSTRATE ABILITY TO APPLY MAGENTO FIXES IN PATCHES

Magento support occasionally provides patches for Commerce customers. When received, place the patch in the `m2-hotfixes/` directory. Once in this directory, test the patch by running `git apply ./m2-hotfixes/{PATCH NAME}` (as you can see it is a git patch). Clear the Magento cache and test. Once satisfied, commit and push. The patch is applied in the build phase.

Ultimately, any third-party vendor can apply patches here, but they may have unintended side effects. `m2-hotfixes` is also useful when third-party modules are installed from repos and need to have bugs or mods made.

It is also used when an emergency fix is needed on production. You can create a patch to fix the issue and add directly to the production branch. You can then fix the problem properly and promote the fix through normal processes.

In addition, `ece-tools` comes with approximately 80 patches to make core Magento code compatible with Magento Cloud. For example, on-demand static-content deploy is not a native Magento feature. This patch is made known to `ece-tools` here, loaded here and applied here.

Further Reading:

Apply custom patches

8.5 DESCRIBE BRANCH SYNCHRONIZATION

Describe branch synchronization and merge

The Project Web Interface and the Magento Cloud CLI provide a toolset to manage environments (aka branches).

Sync

This loads the database and code from a parent branch. In git, there is no concept of hierarchy. `master` is on the same plane as any other branch. Cloud Starter gives more of a three-dimensional perspective: Master > Staging > Integration environments. In Cloud Pro, this is slightly different: Master (`master` branch), Production (what serves content for visitors), Staging > Integration > Integration branches.

With this idea, you can synchronize data from the Master environment back to Staging.

This is done either in the Project Web Interface or the Magento Cloud CLI.

If you are hosting your code in GitHub/GitLab/BitBucket, remember that Magento Cloud is, in effect, its own git repository that is upstream of your repository on GitHub. Performing a synchronization of code is essentially a `git pull origin master` (for example) and merges code. This could cause problems, so it is ideal to do these merges in your primary git repository and don't use merging functionality in Cloud.

Practical experience:

- Synchronize the data on staging from the master environment.

Further Reading:

Sync from the environment's parent

Common Magento Cloud CLI commands

Merge

Please read the above synchronization section as it contains an important note about how this works with using Cloud as an upstream repository from GitHub/GitLab/BitBucket.

The merge function merges code in a child branch with that of the current branch. To promote staging code to production (Starter projects only), merge `staging` into `master`, or `staging` into `production` (Pro projects only). This can happen on the command line, in the Project Web Interface (if you are not using Cloud as an upstream git provider), or in GitHub/GitLab/BitBucket. Note that Magento doesn't handle merge conflicts, so these must be resolved on your local git instance.

9 GO LIVE AND MAINTENANCE

10% of the test

SWIFT OTTER SOLUTIONS

9.1 DEMONSTRATE ABILITY TO CONFIGURE DNS

DNS configuration when you're going live

Before going live, lower the Time-To-Live (TTL) value. This will help the DNS propagate sooner.

Use CNAME records to point to the Fastly DNS endpoint.

It is important to note that you cannot use a CNAME record for the root domain name (for example, `swiftotter.com`). As such, you would need to redirect `swiftotter.com` to `www.swiftotter.com`. You would set up a CNAME record for `www.swiftotter.com` to point to Fastly.

Further Reading:

Switch DNS to the new site

DNS configurations

Why a domain's root can't be a CNAME — and other tidbits about the DNS

9.2 DEMONSTRATE ABILITY TO SET UP AND CONFIGURE FASTLY

Before we discuss Fastly, it is important to make two notes regarding Fastly control panel access:

- If the root domain is hosted in a merchant's Fastly account, you need to create a Fastly support ticket to add Cloud's subdomains in.
- If the root domain is hosted under the Cloud's Fastly implementation, you need to create a Magento support ticket to make your subdomains available for modification in Fastly.

This section will be an overview of the Fastly configuration. Please take the time to thoroughly review the Fastly documentation in DevDocs (see link below) and implement this for yourself in a sandbox environment.

Installing Fastly is easy: just install the `fastly/magento2` Composer package.

Once the module is on Staging, it needs to be configured.

- Enter the correct Fastly API keys (can be found in `/mnt/shared/fastly_tokens.txt`).
- Instruct Magento to use the Fastly CDN as the Caching Application in Store Configuration.
- Upload VCL snippets to Fastly (here are the default ones).
- Configure Fastly to use a Shield location that is closest to the location where the merchant's store is deployed.

You can also configure the timeout for Magento admin pages. Additionally, you can map visitors to specific store views based on their country (IP address).

Further Reading:

Set up Fastly

Fastly Backend Settings

9.3 DEMONSTRATE ABILITY TO UPGRADE TO A NEW VERSION

Upgrade of Magento and ece-tools to newer versions

Upgrading Magento is quite easy. Before running `composer update`, you need to require a range of versions (between two versions). The range is `>=` the desired version and `<` the desired version + one patch version.

To specify Magento 2.3.2, you would use:

```
…
"magento/magento-cloud-metapackage":">=2.3.2 < 2.3.3"
...
```

While the reason for this condition is not stated in Magento documentation, this likely gives Magento the flexibility to release updates to the Cloud meta package (like version 2.3.2.1, for example).

Then, of course, push to integration > staging > production.

Further Reading:

Upgrade Magento version

9.4 DEMONSTRATE ABILITY TO UPSIZE

How to upsize the environment

Pro environments can upsize from 12 CPUs/48GB RAM to 120 CPUs and 480GB RAM. To increase the capacity, you must submit a ticket 72 hours in advance (if you expect a surge of traffic on Monday, you should submit the ticket on at least the Wednesday before). Magento will upsize the server for five days at a time. Support will also upsize automatically if it is necessary to keep the site available (this will still affect contracted upsizing days).

You can also view the upsizing history with the following URL: https://cloud.magento.com/project/{PROJECT ID}/services/clusterresize

The larger the upsize, the more contracted days are used. For example, if the merchant is on an ECE12 plan, and they need to migrate to an ECE24, then you have 45 upsize days available (45 days available for the next level up). If you need to upsize to an ECE48 (2 levels up), then 22 days are available. Note that these can be mixed.

Further Reading:

Pro architecture

Magento Cloud Enterprise Post-Development and Pre-Launch Checklists

Retrospective: The Magento Commerce Cloud at Work