Cisco Intercloud Services

Shipped CLI User Guide

Published 06/07/2016

Copyright Notice

Document History

Topic	Date of Change	Description
Added Change History page	5/26/15	Added a Change History page to the documentation to provide a quick, high-level view of what was modified in the documentation.
Text tables	5/28/15	Text tables have been updated.

About This Document

This document describes the Shipped CLI commands.

Audience

This document is for anyone who uses CLI commands with Shipped.

Related Document

- Shipped Getting Started Guide
- Shipped User Guide
- Shipped API Guide

Conventions

Item	Description
bold	Menu, command.
mono-space font	Code, typed data.
italic OR < >	User input.
	**Note**. Contains information that might be useful. Ignoring a note has no negative consequences.
	**Tip**. Includes information such as helpful hints or a shortcut that might help you complete a task.
	**Important**. Includes information that might be easily overlooked and might cause unnecessary frustration. For example, configuration changes that only apply to the current session, or services that need restarting before an update will apply.
	**Warning**. Contains information that must not be ignored. Ignoring recommendations in Warnings may result in data loss or other catastrophic issues.

Support

To access CCS FAQs and other support resources, or to report support issues, or to open a support request, visit:

https://cisco.jiveon.com/groups/cisco-cloud-services-support/

https://mycloud.telstra.com

For information on CIS, see https://developer.cisco.com/site/cis/partners-alliance/telstra/resources/

Introduction

Shipped is an integrated development, deployment, and operations solution — a centralized location that enables developers to manage all aspects of an application. Developers can authenticate with GitHub, assemble an application with various services and dependencies, and deploy it all to any public or on-premise cloud. Operations teams can monitor and receive alerts from running applications.

Applications Used in Shipped

Shipped automatically runs several OpenStack applications, which includes Docker and Drone.

Docker

Docker allows you to create completely isolated, repeatable, and deployable artifacts. Shipped uses it to run each of your services in isolation. Each service you add to your project has a corresponding Dockerfile that is exposed for you.

Visit Docker for more information.

Drone

Drone provides continuous integration. Shipped adds a web hook into your service repositories; this web hook triggers a build in Drone each time a new commit is made on any branch. Build results show up in Shipped and successful builds show up on the **Deploy** page, ready to release to an Environment. Logs from failed builds are also available from the UI.

The Shipped-CLI bootstrap provides a default *.drone.yml* file for each service. You can edit and commit the .drone.yml file within that service directory.

Visit Drone for more information.

Questions: How do users install CLI?

Authentication

Update

The first time you use the Shipped CLI, you must supply an API token. The token is saved for subsequent logins. Get the token from the Shipped UI: Go to the My Profile page and click Show My User Token.

Command Syntax

The Shipped CLI provides command line access to Shipped services. This includes services that can be performed with the Shipped UI, such as creating and updating projects and services, and deploying them to the cloud. It also provides commands for controlling your local environment, such as bootstrapping projects, building and testing changes.

Shipped CLI commands typically take the form:

```
command subcommand args
```

where
command is a major area of Shipped (such as project, service, or build),
subcommand is what you want to do (such as create or list), and
args are additional arguments used by the command.

For example,

```
shipped build get 183d5961-71f5-11e5-b058-0242ac110238
```

Help

The **help** command gives you access to the built-in help. Type `shipped -h` or `shipped --help` to get a list of commands.

Some common commands are:

Command	Description
`shipped help`	Overview of the CLI (same information as this page)
`shipped help` *command*	Help on command *command*
`shipped help` *command subcommand*	Help on *subcommand* of *command*
`shipped help workflow`	Overview of Shipped orchestrated workflow
`shipped wf help` *workflowName*	Help on individual workflow script *workflowName*

Command	Description
`shipped help default`	Context sensitivity and default values features

Note to Doris: Need to move the Options section

Options

CLI		Description
-b	--bootlogfile	Name of file to write bootstrap log messages [default shipped-bootstrap.log].
-d	--debug	Provide TRACE logging.
-f	--full	Show full model (all elements) in command responses.
-h	--help	Display help message for current command.
-j	--json	Output command responses in JSON format.
-l	--logfile	Name of file to write log messages, including TRACE messages (also SHIPPED_CLI_LOGFILE) [default shipped-cli.log].
-o	--output	Comma-separated list of output field names.
-p	--project	Home directory for Shipped projects (also SHIPPED_PROJECT_HOME). When this option is provided, Shipped creates project directories as a subdirectory of SHIPPED_PROJECT_HOME, and automatically changes to this directory as needed.
-x	--expanded	Include expanded descriptions in wizard workflow.
-s	--server	API server to access (also SHIPPED_API_SERVER) [default http://api.ciscoshipped.io].
-t	--token	Shipped API authentication token (also SHIPPED_API_TOKEN).
-v	--version	Show Shipped CLI version and exit.

Shipped CLI Commands

The following lists and describes the top-level commands and their abbreviations (if available). You can abbreviate commands, subcommands, and arguments by truncation to the minimum length necessary for unambiguous specification.

Shipped Command	Abbreviation	Description
Application	app	Applications have multiple microservices and backends that are wired together for building containerized microservices.
Build	bld	Builds pipeline for each service created within an application.
BuildPack	bp	Provides the bare-bones code for developing a service
Config	cfg	A configuration includes instructions to run a service.
Console		Enters the console mode.
Defaults	dflt	Get the default values.
environments	env	An environment is a deployable configuration set for your project.
Local	lcl	Local manages the local (desktop) development environment.
Releases	rlse	A release is a specific build deployed to a specific environment for a service.
Run		Runs a Shipped workflow.
Services	svc	A Shipped service represents a microservice within an application that is independently deployable and buildable.
Users	usr	Shipped supports multiple users for application collaboration.
Wizard	wz	Provides a basic tutorial to the Shipped CLI and walks you through a task workflow.
Workflow	wf	Carries out an orchestrated and repeatable set of commands that performs a high-level operation.

Application

Shipped is best suited to building containerized microservices based applications. Application typically have multiple microservices and backends that are wired together. An application can have its own independent development and deployment life cycle.

The following table lists and describes available subcommands of application.

Application Subcommand	Description	Usage
create	Create an application.	`application create <name=> [<description=>]`
delete	Delete an application.	`application delete <projectId>`
get	Return details for one or all applications.	`application get <projectId>`
getall	Return a list of all applications.	`application getall`
update	Update an application.	`application update update <projectId> <name=> [description=]`

application create

Create an application.

Usage

```
application create <name=> [<description=>]
```

Argument and Option

Argument and Option	Description
<name=>	Required. The name for the application. Enclose the name in quotes as in "TestProject."
description=	Optional. A description for the application. Enclose the description in quotes as in "A sample project."

Command Example

```
shipped app create name="TestProject" Description="A sample project"
```

Return

```
Name         ProjectID                            Description
----         ---------                            -----------
TestProject  34601eaa-6c5f-11e5-97ca-0242ac11063c  A sample project
```

application delete

Delete a project and the associated data, including services and deployment environments.

Usage

```
application delete <projectId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project to delete.

Command Example

```
shipped app delete %demo
```

Return

```
Are you sure you want to delete project demo-api-issue? [y]
```

application get

Return details for one or all applications. Use the getall subcommand if you are invoking the CLI from a project directory.

Usage

```
application get <projectId>
```

Argument and Option

Argument and Option	Description
<projectId>	Optional. The ID of the project to return. If the project ID is not specified, all applications are returned.

Command Example

```
shipped project get 183d5961-71f5-11e5-b058-0242ac110238
```

Return Example

```
Name        ProjectID                            Description
----        ---------                            -----------
godemo-new  183d5961-71f5-11e5-b058-0242ac110238  New Go project
```

application getall

Return details for all applications. Getall is useful if you are invoking the CLI from a project directory.

Usage

```
application getall
```

Argument and Option

None

Command Example

```
shipped project getall
```

Return Example

```
shipped getall

Name       ProjectID                              Description
----       ---------                              -----------
godemo-new  183d5961-71f5-11e5-b058-0242ac110238  New Go project
kanboard    7e7e2b43-6861-11e5-851d-0242ac113ce8
```

application update

Update an application description.

Question: Can the name be updated?

Usage

```
application update update <projectId> <name=> [description=]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the application to update
<name=>	Required. The name for the application.
description=	Optional. The updated description for the application. Enclose the description in quotes as in "updated application."

Command Example

```
shipped application upd name="TestProject" desc="updated application"
```

Return Example

```
id                                   name         description         last_seen_event_id  event_count  user_count
--                                   ----         -------------       ------------------  -----------  ----------
f10e6237-72bf-11e5-9202-0242ac11026c TestProject  updated application                       0            0
```

Build

Shipped auto-configures a continuous integration (CI) build pipeline for each service created within project. Shipped also (upon creation of a service) registers a callback hook in GitHub so every commit triggers a new build in the CI pipeline. The build job typically involves executing compiling source code, running unit test, static code analysis, code coverage, creating Docker image, and publishing a Docker image.

Every commit on a GitHub repo triggers a new build and upon successful completion of a build new Docker image tagged with a commit ID is pushed to a project-specific Docker private repository. Modify the .drone.yml config file in the base directory of the source repository of the service to customize the CI build script.

The following table lists and describes available subcommands of build.

Build Subcommand	Description	Usage
deploy	Deploy a project service specific build to a project deployment environment.	`build deploy <commitId> <environmentId> <projectId> <serviceId> [optional arguments]`
get	Return build details for one service or all services in a project.	`build get <projectId> [<serviceId>] [--latest]`
getlogs	Return logs for a specific service build.	`build getlogs <projectId> <serviceId> <buildId>`
set	Set the default build ID and commit ID in a service's CLI context.	`build set <projectId> <serviceId> [<buildId>] [--latest]`

build deploy

Deploy a project service specific build to a project deployment environment. Each build creates a unique Docker image tagged with commit ID in a Docker private repository. The deploy subcommand deploys a specific build image for a service. It sets the releaseId of the new release in the CLI context for the service, so subsequent commands (such as rlse get) default to fetching that release.

Usage

```
Usage: build deploy <commitId> <environmentId> <projectId> <serviceId> [optional arguments]
```

Exercise caution when using optional arguments. Do not change the default values unless you have a good understanding of Shipped and container orchestration. For less experience users, you can use the build deployment form provided on the Shipped Web UI to customize optional arguments.

Argument and Option

Argument and Option	Description
<commitId>	Required. The commit ID of the build to release.
<environmentId>	Required. The environment ID of the environment where the build should be released.
<projectId>	Required. The project ID of the project owing the service that was built.
<serviceId>	Required. The service ID of the service owning the build.
ConfigID=	Optional. ConfigID to be deployed.
ContainerCount=	Optional. Number of service containers to deploy.
ContainerCPU=	Optional. vCPU to allocate per container.
ContainerPort=	Optional. Port on which app running inside container.
ContainerRAM=	Optional. RAM in MB to be allocated for container.
DeployedURL=	Optional. Where the service is or would be running.
DeployImage=	Optional. The Docker image that got deployed.
DeploymentId=	Optional. Recommended default set by Shipped.

Argument and Option	Description
EnvName=	Optional. Recommended default set by Shipped.
EnvVariables=	Optional. Recommended default set by Shipped.
ErrorMsg=	Optional. Populated if there was a problem.
HealthCheckMaxFailures=	Optional. Health check failure before marking service as down, default by buildpack.
HealthCheckPath=	Optional. Health check endpoint URL, default by buildpack.
HealthCheckProtocol=	Optional. Health check protocol, default by buildpack.
MarathonAppID=	Optional. The marathon identifier for the app.
RepoURL=	Optional. The GitHub repo for the app.
ServiceType=	Optional. App, RepoApp, or Service.
SnapshotID=	Optional. If multiple services are deployed from a single deploy, they all have the same snapshotID. Useful for rollbacks.

Example -- deploy process

1. Issue **build get** from outside bootstrapped local development environment.

```
shipped build get 8f005490-6ada-11e5-9686-0242ac11063c 9367df79-6ada-11e5-9687-0242ac11063c BuildID
Timestamp Status CommitID

49260b7d-6afb-11e5-969a-0242ac11063c 2015-10-05 00:52:41 +0000 UTC Success
2687a3cba0a904e1c4ef91185557bc9dd78b8ca5 2ef3977b-6ae7-11e5-9692-0242ac11063c 2015-10-04 22:28:59
+0000 UTC Success 16144e192f4fc3eb0ed5553f436609ed3676206a 5b39921f-6adb-11e5-968c-0242ac11063c
2015-10-04 21:03:44 +0000 UTC Failure dbf4e28fe231ecc9bddc38a4706a81f492eaf5dc f0144ecd-
6ada-11e5-9689-0242ac11063c 2015-10-04 21:03:48 +0000 UTC Success
35a11dc94a320da32b38674483d94f90622af909
```

   Since the command is executed outside the development environment, projectid and serviceid need to be passed explicitly to the **build get** command. However, Shipped automatically detects these IDs if the directory is changed to the service directory.

2. **cd** to the projectname/sevicename directory in the bootstrapped environment.

Shipped CLI automatically detects both projecidt and serviceid; therefore, you do not need to enter these parameters while making calls.

```
cd demoproject/demoservice
shipped build get
BuildID                                Timestamp                       Status   CommitID
-------                                ---------                       ------   --------
49260b7d-6afb-11e5-969a-0242ac11063c   2015-10-05 00:52:41 +0000 UTC   Success  2687a3cba0a904e1c4ef91185557bc9dd78b8ca5
2ef3977b-6ae7-11e5-9692-0242ac11063c   2015-10-04 22:28:59 +0000 UTC   Success  16144e192f4fc3eb0ed5553f436609ed3676206a
5b39921f-6adb-11e5-968c-0242ac11063c   2015-10-04 21:03:44 +0000 UTC   Failure  dbf4e28fe231ecc9bddc38a4706a81f492eaf5dc
f0144ecd-6ada-11e5-9689-0242ac11063c   2015-10-04 21:03:48 +0000 UTC   Success  35a11dc94a320da32b38674483d94f90622af909
```

3. Issue **build getlogs** to fetch logs for a specific build.

```
shipped build getlogs $ $ 2ef
$ git clone --depth=50 --recursive --branch=master git://github.com/npateriya/wfservice5.git
/var/cache/drone/src/github.com/npateriya/wfservice5
Cloning into '/var/cache/drone/src/github.com/npateriya/wfservice5'...
$ git checkout -qf 16144e192f4fc3eb0ed5553f436609ed3676206a
$ echo "build commands"
build commands
....
```

The shipped command is issued from the service directory, allowing `projectid` and `serviceid` to be passed as $, indicating that they should take the default values. The `buildid` parameter contains only the first few characters that uniquely identify it, and which the CLI expands internally.

Use dollar sign ($) in the argument position to use the default value; each $ represents one default.

You only need enough leading characters of the UUID to uniquely identify the buildid.

4. Issue **build set** to set the specific buildId as default that will be used for subsequent calls to Shipped CLI.

```
shipped build set $ 2ef
Setting project wfproj5 default buildId(2ef3977b-6ae7-11e5-9692-0242ac11063c) and commitId(16144e192f4fc3eb0ed5553f436609ed3676206a)
```

5. Issue **build deploy** to deploy the build to the environment.

   The release ID is returned.

```
shipped build deploy
EnvName  ReleaseID                              ProjectName  ServiceName  Status  ErrorMsg  DeployedURL
-------  ---------                              -----------  -----------  ------  --------  -----------
test     e578c196-6b9b-11e5-96fe-0242ac11063c   wfproj5      wfservice5
```

build get

Return build details for one or all services in a project.

Note to Doris: project must be built first.

Usage

```
build get <projectId> [<serviceId>] [--latest]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to get build details.
serviceId	Optional. The ID of service for which to get build details. If not specified, all service IDs are returned.
Option	
--latest	Return details only on the most recent build.

Command Example

```
shipped build get 183d5961-71f5-11e5-b058-0242ac110238
```

Return Example

```
shipped build get 183d5961-71f5-11e5-b058-0242ac110238
BuildID                              Timestamp                      Status   CommitID
-------                              ---------                      ------   --------
49260b7d-6afb-11e5-969a-0242ac11063c  2015-10-05 00:52:41 +0000 UTC  Success  2687a3cba0a904e1c4ef91185557bc9dd78b8ca5
2ef3977b-6ae7-11e5-9692-0242ac11063c  2015-10-04 22:28:59 +0000 UTC  Success  16144e192f4fc3eb0ed5553f436609ed3676206a
5b39921f-6adb-11e5-968c-0242ac11063c  2015-10-04 21:03:44 +0000 UTC  Failure  dbf4e28fe231ecc9bddc38a4706a81f492eaf5dc
f0144ecd-6ada-11e5-9689-0242ac11063c  2015-10-04 21:03:48 +0000 UTC  Success  35a11dc94a320da32b38674483d94f90622af909
```

build getlogs

Return logs for a specific service build. Logs are `stdout` and `stderr` captured by CI server during execution of CI scripts specified in .drone.yml.

Currently, `getlogs` is limited to logs that are available when the build is complete.

Usage

```
build getlogs <projectId> <serviceId> <buildId>
```

Argument and Option

need xref

Argument and Option	Description
<projectId>	Required. The ID of the project for which to return the logs.
<serviceId>	Required. The ID of service for which to return the logs.
<buildId>	Required. The ID of build for which to return the logs.

Command Example

```
shipped build getlogs 183d5961-71f5-11e5-b058-0242ac110238 271c696b-7360-11e5-9240-0242ac11026c 49260b7d-6afb-11e5-969a-0242ac11063c
```

Return Example

Need example

build set

Set the default build ID and commit ID in a service's CLI context. This command determines the commit ID from a project, service and build. This allows you to skip explicitly passing the ID in subsequent CLI calls.

Usage

```
build set <projectId> <serviceId> [<buildId>] [--latest]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project that owns the build. **Question: Should this be optional when --latest is used?**
<serviceId>	Required. The ID of the service that is built by the build.
<buildId>	Required. The build ID to be set as the default.
Option	
--latest	Set the build ID to that of the most recent build (and ignore the <buildId> argument).

Command Example

```
shipped build set 183d5961-71f5-11e5-b058-0242ac110238 271c696b-7360-11e5-9240-0242ac11026c 49260b7d-
6afb-11e5-969a-0242ac11063c
```

Return

OK.

BuildPack

Shipped offers a pre-built build package called BuildPack. A BuildPack provides the bare-bones code for developing a service. For example, if you are building a Ruby on Rails app, a BuildPack would set up a working, deployable HelloWorld application with appropriate defaults. Using BuildPacks as starting points for your new applications saves time and simplifies the process.

Choosing a BuildPack automatically configures the default values for the following key areas in Shipped. Each of these can be customized as needed.

- **Build**: The BuildPack sets up a build and test environment for the selected programming language. Required build/test tools for selected languages are installed with the build container. Default commands of building and testing service code is also auto configured.
- **Runtime**: Depending on the language and framework of the selected BuildPack, the runtime base container with installed runtime container is automatically configured. This also sets up other runtime configurations, such as exposing the default port. This can be customized by changing the FROM container in the Dockerfile service.
- **Sample Application**: Each BuildPack automatically populates a working HelloWorld sample code when creating a new GitHub repo. If a service is created for an existing repo, then the sample creation is skipped and the config files are added to the project folder.

The following table lists and describes available subcommands of BuildPack.

BuildPack Subcommand	Description	Usage
get	Returns a specific BuildPack when a BuildPack ID is specified; otherwise, all BuildPacks are returned.	`buildpack get [<buildpackId>]`
getall	Returns a list of available BuildPacks .	`buildpack getall`
set	Sets the default BuildPack ID.	`buildpack set`

buildpack get

Return one BuildPack or all BuildPacks. To get a list of available BuildPacks, issue the getall command.

Usage

```
buildpack get [<buildpackId>]
```

Argument and Option

Argument and Option	Description
buildpackId	Optional. If the ID of the BuildPack is not specified, all BuildPacks are returned.
Option	
--full	Return details of BuildPack fields.
--json	Return information in JSON format.

Command Example

```
shipped --json --full buildpack get 1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29
```

Return Example 1 -- returns information for a specific BuildPack

```
shipped buildpack get 1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29

Name        BuildPackID
----        -----------
Dropwizard  1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29
```

Return Example 2 -- returns a specific BuildPack in JSON format

```
shipped --json --full buildpack get 1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29
{
    "id": "1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29",
    "name": "Dropwizard",
    "image_source": "http://shipped-api.shipped-cisco.com/cli/static/images/dropwizard.svg",
    "default_build_command": "/app/bin/build",
    "default_run_command": "/app/bin/run",
    "default_test_command": "/app/bin/test",
    "default_cpu": 1,
    "default_ram": 512,
    "build_image": "jamesdbloom/docker-java8-maven",
    "default_deploy_image": "jamesdbloom/docker-java8-maven",
    "default_container_port": 8080,
    "container_shared_directory": "/app",
    "bootstrap_template_repo": "http://github.com/CiscoCloud/shipped-buildpack-template-dropwizard.git",
    "default_health_check_path": "/"
}
```

buildpack getall

Return a list of available BuildPacks. To get all BuildPacks, issue the get command.

Usage

```
buildpack getall
```

Argument and Option

None

Command Example

```
shipped buildpack getall
```

Return Example

```
shipped buildpack getall

Name           BuildPackID
----           -----------
Dropwizard     1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29
ExpressJS      1da3b58c-0b3b-11e5-9cb7-7cd1c3e98f29
golang         1d364ebd-0b3b-11e5-9cb3-7cd1c3e98f29
Ruby on Rails  1d6cf3c3-0b3b-11e5-9cb5-7cd1c3e98f29
Python         1d51c938-0b3b-11e5-9cb4-7cd1c3e98f29
ASP.net MVC    1dbeda73-0b3b-11e5-9cb8-7cd1c3e98f29
APICEM         0d296ec9-403a-11e5-bc38-0242ac1106ab
Grunt.JS       1d88a414-0b3b-11e5-9cb6-7cd1c3e98f29
CiscoCMX API   1df60fc4-0b3b-11e5-9cba-7cd1c3e98f29
PHP            f5e20759-f2b9-11e4-963b-14109fd75217
```

buildpack set

Set the default BuildPack ID.

Question: Why would users want to do this?

Usage

```
buildpack set <buildpackId>
```

Argument and Option

Argument and Option	Description
buildpackId	Required. The ID of the BuildPack to set as the default.

Command Example

```
shipped buildpack set 1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29
```

Return

OK

Config

A configuration contains all the instructions to run a service. A service has many configurations, at least one per environment in your project. Each configuration also has an associated deploy target.

The following table lists and describes available subcommands of config.

Config Subcommand	Description	Usage
delete	Delete a service configuration.	`config delete <projectId> <serviceId> <environmentId> <configId>`
getall	Return all configurations.	`config getall <projectId> <serviceId> <environmentId>`
set	Set the default config ID.	`config set <configId>`
update	Update an environment configuration.	`config update <projectId> <serviceId> <environmentId> <configId> [<optionalArgs>]`

config delete

Delete a service configuration.

Usage

```
config delete <projectId> <serviceId> <environmentId> <configId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project in which the configuration you want to delete.
<serviceId>	Required. The ID of the service in which the configuration you want to delete.
<environmentId>	Required. The ID of the environment in which the configuration you want to delete.
<configId>	Required. The ID of config you want to delete

Command Example

```
shipped config delete 183d5961-71f5-11e5-b058-0242ac110238 271c696b-7360-11e5-9240-0242ac11026c 49260b7d-
6afb-11e5-969a-0242ac11063c b1d59858-6d57-11e5-aa3e-0242ac115798
```

Return

Returns OK if successful.

config getall

Return all configurations.

Usage

```
config getall <projectId> <serviceId> <environmentId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project in which the configuration you want to retrieve.
<serviceId>	Required. The ID of the service in which the configuration you want to retrieve.
<environmentId>	Required. The ID of the environment in which the configuration you want to retrieve.

Command Example

```
shipped cfg getall 183d5961-71f5-11e5-b058-0242ac110238 271c696b-7360-11e5-9240-0242ac11026c 49260b7d-
6afb-11e5-969a-0242ac11063c
```

Return Example

```
shipped cfg getall
ConfigID                              ProjectName  ServiceName    EnvName
--------                              -----------  -----------    -------
b1d59858-6d57-11e5-aa3e-0242ac115798  test-1007    test-1007-svc  staging
```

config set

Set the default config ID.

Usage

```
config set <configId>
```

Argument and Option

Argument and Option	Description
<configId>	Required. The ID of config you want to set as the default.

Command Example

```
shipped cfg set b1d59858-6d57-11e5-aa3e-0242ac115798
```

Return

Returns OK if successful.

config update

Update an environment configuration.

Usage

```
config update <projectId> <serviceId> <environmentId> <configId> [<optionalArgs>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project in which the configuration you want to update.
<serviceId>	Required. The ID of the service in which the configuration you want to update.
<environmentId>	Required. The ID of the environment in which the configuration you want to update.
<configId>	Required. The ID of config in which the configuration you want to update.
ContainerCount=	Optional. The number of containers that Marathon will deploy in this environment.
ContainerCPU=	Optional. The CPU that Marathon will allocate for deployments to this environment.
ContainerPort=	Optional. The port number that Marathon will use for deployments to this environment.
ContainerRAM=	Optional. The RAM that Marathon will allocate for deployments to this environment. **Question: RAM in GB?**
Deployimage=	Optional. The image (usually found at registry.hub.docker.com) of the docker container in which to run. The name of the image must be enclosed in quotes, for example, deployimage="GP2-Large/nodejs-ssh:latest".

Command Example

```
shipped cfg upd 183d5961-71f5-11e5-b058-0242ac110238 271c696b-7360-11e5-9240-0242ac11026c 49260b7d-6afb-
11e5-969a-0242ac11063c b1d59858-6d57-11e5-aa3e-0242ac115798 containerram=128 containerport=3000
containercount=1 containercpu=0.5 deployimage="GP2-Large/nodejs-ssh:latest"
```

Return

Returns OK if successful.

Console

You can use the Shipped CLI in console mode where Shipped prompts for commands and supports its own command history and command completion. To enter console mode, issue **shipped console**, for example,

```
demo@ubuntu:~/shipped: shipped console
Cisco Shipped 1.0.  Type help for help.
Shipped>
```

To exit console mode, enter **exit**, **quit**, or **q**.

When you are in the console mode:

- Do not use the "shipped" prefix on commands.
- You can use the Up and Down arrow keys to scroll through a history of previous commands.
- You can press the Tab key to complete command names and arguments.
- You can use the **resume** or **wf resume** command to continue execution of a failed workflow at the point of the failure.
- You can use Docker-style or %xxx references to extract UUIDs from the preceding command output. This lets you supply ID values for objects other than builds, buildpacks, environments, projects, and services.

Defaults

Many commands use one or more positional arguments that are usually UUIDs. Use the **service get** command as an example:

```
shipped service get <projectId> [<serviceId>]
```

The command has two arguments, a required argument of a project ID and an optional argument of a serviceID. To supply these arguments:

- Invoke the CLI from a project or service directory, causing the project ID argument to default to the ID of the project. If you invoke the CLI from a service directory, the service ID is the default. This is Shipped CLI context sensitivity.
- Supply the first few characters of the UUID, Docker-style, for either argument. For example, if the project ID is "c272e20d-6ab4-11e5-967e-0242ac11063c," simply specify **svc get c27**. You only need to specify enough leading characters to uniquely identify the project.
- Supply a portion of the name of the project or service, preceded by a percent (%) symbol. For example, the project is named "Hello-World" and the service "my-php-service," you can specify **svc get %wor %php**. Note that a name specification is not case sensitive and can be any part of the name (not necessarily the beginning). You only need to specify enough characters to uniquely identify the project or service.

The following table lists and describes available Defaults subcommands.

Defaults Subcommand	Description	Usage
get	Return the default values active in the current context.	`defaults get`

defaults get

Return the default values active in the current context. Shipped is context-sensitive, so the values listed for this command vary depending on whether it is issued from a Shipped project or service directory.

Usage

```
defaults get
```

Argument and Option

None

Command Example

This example shows the default is issued from a service directory.

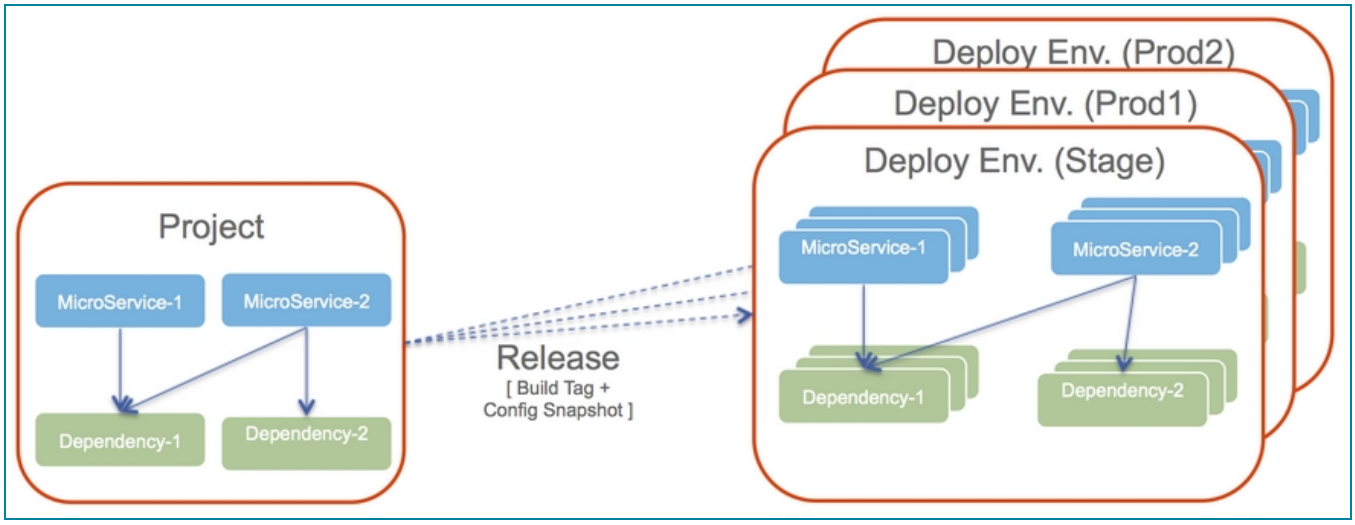
```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped defaults get
```

Return Example

```
name           value                                  description
----           -----                                  -----------
apiToken       RLUtxxxxxxxxxxxxxxxxxxxxxxxxxxIub
environmentId  1066e5b9-6ba1-11e5-970b-0242ac11063c
projectId      c272e20d-6ab4-11e5-967e-0242ac11063c   test-CI
projectName    test-CI
serviceId      c9f2e8fc-6ab4-11e5-967f-0242ac11063c   test-ci-svc
serviceName    test-ci-svc
```

Environment

Shipped supports an isolated namespace for deploying projects. An environment is a deployable configuration set for your project. For example, it is typical to have a Production environment and a Staging environment operating at very different scales. You can create multiple environments as needed. Each environment contains a specific configuration for each service in your application that defines global parameters for your application.



Shipped supports auto-discovery of services within the same environment. Shipped also automatically sets up an internal load balancer for instances of each service with unique endpoint per environment. The service URL endpoint format is:

<envname>--<projectname>--<servicename>.<cluster_sudomain>

For example:

```
stage--demoproject--demoservice.shipped-cisco.com
```

The following table lists and describes available subcommands of environment.

Environment Subcommand	Description	Usage
create	Create a project environment.	`environment create <projectId> name= [description=]`
delete	Create a complete project environment.	`environment delete <projectId> <environmentId>`
get	Return one or all project environments.	`environment get <projectId> [<environmentId>]`
getall	Return all environments for a project.	`environment getall <projectId>`
set	Set the default environment ID.	`environment set <environmentId>`

Environment create

Create a project environment. Create environment internally triggers creation of a configuration for every service in the project with a default deploy target.

Usage

```
environment create <projectId> name= [description=]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to create the environment.
<name=>	Required. The name for the environment.
description=	Optional. A description for the environment. Enclose the description in quotes as in "environment to deploy test cluster."

Command Example

```
shipped env create 8f005490-6ada-11e5-9686-0242ac11063c name=test description="environment to deploy test
cluster"
```

Return

```
project_id                           id                                   name  description
----------                           --                                   ----  -----------
8f005490-6ada-11e5-9686-0242ac11063c 11b81092-6b9b-11e5-96f9-0242ac11063c test  Enviroment to deploy test cluster
```

environment delete

Delete a project environment. Delete environment internally also cleans up the deployed container for services in the project for the specified environment. For example, deleting the "Production" environment deletes the namespace and the running containers for this environment.

Use this option only when you want to delete the complete deployment environment.

Usage

```
environment delete <projectId> <environmentId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to delete the environment.
<environmentId>	Required. The ID of the environment to delete

Command Example

```
shipped env delete 8f005490-6ada-11e5-9686-0242ac11063c 271c696b-7360-11e5-9240-0242ac11026c
```

Return

OK if successful.

environment get

Return one or all project environments.

Usage

```
environment get <projectId> [<environmentId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to list the environment.
environmentId	Optional. The ID of the environment to return. If no environment ID is specified, all environment IDs are returned.

Command Example

```
shipped env get 8f005490-6ada-11e5-9686-0242ac11063c 271c696b-7360-11e5-9240-0242ac11026c
```

Return

Question: Need example

environment getall

Return all environments for a project.

Usage

```
environment getall <projectId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to list the environment.

Command Example

```
shipped env getall 8f005490-6ada-11e5-9686-0242ac11063c
```

Return

Question: need example

environment set

Set the default environment ID.

Usage

```
environment set <environmentId>
```

Argument and Option

Argument and Option	Description
<environmentId>	Required. The ID of the environment to set as the default.

Command Example

```
shipped env set 271c696b-7360-11e5-9240-0242ac11026c
```

Return

OK if successful.

Local (lcl)

Local manages the local (desktop) development environment.

Each service in a project runs in its own Docker container, with the containers hosted in a Docker host VM. Each container includes everything needed to build and run the service, so there is no need to install service-related software (such as compilers) on your laptop. A service's local source repository is shared with its container, so that changes you make on your desktop are available to the container. You build, run, and test your code in the container, while maintaining it on your desktop. In addition, for most services, the container hosts a Web server with a port exposed on your laptop allowing you to test your service's Web UI on your laptop.

Shipped automatically sets up a VM for the project and its services when you perform a normal (that is, not a fast) bootstrap for the project. If you add a new service to a project, or originally ran a fast bootstrap, you need to run bootstrap again before using most of the local commands.

The following table lists and describes available subcommands of local.

local Subcommand	Description	Usage
bootstrap	Bootstrap a Shipped project.	`lcl bootstrap <projectId> [--fast|--localDocker]`
build	Build a service in its container.	`lcl build <projectId> <serviceId>`
commit	Commit changes to a Shipped project and starts a CI build.	`lcl commit <projectId> [<serviceId>] message=`
destroyVM	Destroy the VM running project services.	`lcl DestroyVM <projectId> [<serviceId>]`
haltVM	Stop the VM running project services.	`lcl haltvm <projectId> [<serviceId>]`

local Subcommand	Description	Usage
logs	Return log files.	`lcl logs <projectId> [<serviceId>]`
reloadVM	stop and restart the VM running project services.	`lcl reloadvm <projectId> [<serviceId>]`
statusVM	Return the status of the project's VM and its service containers.	`lcl statusvm <projectId> [<serviceId>]`
test	Test a service in its container.	`lcl test <projectId> <serviceId>`

local bootstrap

Bootstrap a Shipped project.

Bootstrapping a project sets it up on your local laptop by:

- Setting up a local Git repository for each of the project's services and cloning the remote GitHub repository set up by Shipped.
- Creating and starting a VM with a Docker container for each of the project's services

Usage

```
lcl bootstrap <projectId> [--fast|--localDocker]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project to bootstrap.
--fast	Optional. Speed up the time to bootstrap the project by skipping both prequisite software installation and VM creation. Using this flag disables the following local features: • local build • local commit • local text To make them available, run bootstrap again without the --fast flag.
--localDocker	Optional. Use Docker on localhost for service container deployment. localDocker is ignored when --fast is specified.  You need Docker installed in your local laptop.

Command Example

```
shipped local bootstrap 8f005490-6ada-11e5-9686-0242ac11063c
```

Question: Need example

local build

Build a service in its container. Compiles and builds a service in its container by running script bin/build (stored in the local repository but run in the container). Since this script runs in the service's container, it uses the support software (such as compilers) automatically provided in the container. The local GitHub repository is synced with the container, so you can use this command after updating source files to build with the changes and prepare them for local testing.

This command is available only after a full bootstrap.

Usage

```
lcl build <projectId> <serviceId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project to build.
<serviceId>	Required. The ID of the service to build.

Command Example

```
shipped local build 8f005490-6ada-11e5-9686-0242ac11063c 271c696b-7360-11e5-9240-0242ac11026c
```

Return Example

Question: Need example

local commit

Commit changes to a Shipped project and starts a CI build. This is a convenience command that performs a Git commit followed by a Git push to the master. It takes changes from your local repository and stores them in the remote GitHub repository created by Shipped. After the commit completes, Shipped automatically starts a CI build for the service.

Usage

```
lcl commit <projectId> [<serviceId>] message=
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project to commit.
serviceId	Optional. The ID of the service to commit. If no service ID is specified, all services are committed.
message=	Required. A description for the commit. Enclosed the message in quotes.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl commit m="First commit"
Commit for service hello-world-cmx (2c916d72-6c2d-11e5-97a6-0242ac11063c): git commit --allow-empty -m 'First commit'
[master 8f0c2ec] First commit
Push for service hello-world-cmx (2c916d72-6c2d-11e5-97a6-0242ac11063c): git push origin master
To https://github.com/tooda02/hello-world-cmx
 * [new branch]      master -> master
```

local DestroyVM

Destroys the VM running project services. This command removes all files associated with the VM from your laptop. After it completes, you will be unable to build and test services locally until you recreate the VM with the local bootstrap command.

Usage

```
lcl DestroyVM <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to destroy service VMs.
serviceId	Optional. The ID of the service for which to destroy service VMs. If no service ID is specified, all service VMs are destroyed.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl destroy
==> hello_world_cmx: Stopping container...
==> hello_world_cmx: Deleting the container...
==> hello_world_cmx: Removing built image...
==> hello_world_cmx: Removing synced folders...
```

local HaltVM

This command stops the VM but leaves its files intact so that it can be restarted quickly. After it completes, you will be unable to build and test services locally until you restart the VM with either the local reloadVM command or the local bootstrap command.

Usage

```
lcl haltvm <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to stop service VMs.
serviceId	Optional. The ID of the service for which to stop service VMs. If no service ID is specified, all service VMs are stopped.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped local halt
==> hello_world_cmx: Stopping container...
```

local logs

Return log files that a service has written to its container.

Usage

```
lcl logs <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to return service logs.
serviceId	Optional. The ID of the service for which to return service logs. If no service ID is specified, all service logs are returned.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl logs
==> hello_world_cmx: serve-static@1.9.3 node_modules/serve-static
==> hello_world_cmx: â"œâ"€â"€ utils-merge@1.0.0
==> hello_world_cmx: â"œâ"€â"€ escape-html@1.0.1
==> hello_world_cmx: â"œâ"€â"€ parseurl@1.3.0
==> hello_world_cmx: â"″â"€â"€ send@0.12.3 (destroy@1.0.3, fresh@0.2.4, range-parser@1.0.2, ms@0.7.1, depd@1.0.1, debug@2.2.0, mime@1.3.4,
on-finished@2.2.1, etag@1.6.0)
==> hello_world_cmx: compression@1.4.4 node_modules/compression
==> hello_world_cmx: â"œâ"€â"€ vary@1.0.1
...
==> hello_world_cmx: â"œâ"€â"€ async@0.9.2
==> hello_world_cmx: â"″â"€â"€ yargs@3.15.0 (decamelize@1.0.0, camelcase@1.2.1, window-size@0.1.2, cliui@2.1.0)
==> hello_world_cmx: Server is listening on port 3000 and virtualizing the CMX server with local file data
```

local ReloadVM

This command recycles the VM, including restarting any Web server resident on the VM.

Usage

```
lcl reloadvm <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to reload service VMs.
serviceId	Optional. The ID of the service for which to reload service VMs. If no service ID is specified, all service VMs are reloaded.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl reloadvm
==> hello_world_cmx: Stopping container...
==> hello_world_cmx: Deleting the container...
==> hello_world_cmx: Removing built image...
==> hello_world_cmx: Removing synced folders...
==> hello_world_cmx: Syncing folders to the host VM...
    hello_world_cmx: Mounting shared folders...
    hello_world_cmx: /var/lib/docker/docker_1444143483_61136 => /home/demo/shipped/hello-world/hello_world_cmx
    hello_world_cmx: /var/lib/docker/docker_1444143483_22891 => /home/demo/shipped/hello-world/.shipped
...
==> hello_world_cmx: Creating the container...
    hello_world_cmx:   Name: hello_world_cmx
    hello_world_cmx:  Image: 5660e37303b5
    hello_world_cmx: Volume: /var/lib/docker/docker_1444143483_61136:/app
    hello_world_cmx: Volume: /var/lib/docker/docker_1444143483_22891:/vagrant
    hello_world_cmx:   Port: 33000:3000
    hello_world_cmx:
    hello_world_cmx: Container created: 9deea14bd80d7687
==> hello_world_cmx: Starting container...
==> hello_world_cmx: Provisioners will not be run since container doesn't support SSH.
```

local StatusVM

Return the status of the project's VM and its service containers. This command displays information about one or all of the containers running project services, including whether they are running and the local port where the container's web server is available.

Usage

```
lcl statusvm <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project for which to return the status of service VMs.
serviceId	Optional. The ID of the service for which to return the status of service VMs. If no service ID is specified, all status of service VMs are returned.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped local statusVM

1 service is running and is available at the web address below
------------     -------          ------   -------------------
Service Name     VM Name          Status   Service Web Address
------------     -------          ------   -------------------
hello-world-cmx  hello_world_cmx  running  http://localhost:33000
```

local test

Tests a service in its container by running script bin/test (stored in the local repository but run in the container). The local GitHub repository is synced with the container, so you can use this command after updating source files and running shipped local build. This command is available only after a full bootstrap.

Usage

```
lcl test <projectId> <serviceId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project containing service to test.
<serviceId>	Required. The ID of the service to test.

Example

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped local test
==> hello_world_cmx: Docker host is required. One will be created if necessary...
    hello_world_cmx: Docker host VM is already ready.
==> hello_world_cmx: Image is already built from the Dockerfile. `vagrant reload` to rebuild.
==> hello_world_cmx: Creating the container...
    hello_world_cmx:   Name: hello_world_cmx_1444143207
    hello_world_cmx:  Image: 85a3b893a4cc
    hello_world_cmx:    Cmd: /bin/bash bin/test
    hello_world_cmx: Volume: /var/lib/docker/docker_1444137749_77123:/app
    hello_world_cmx: Volume: /var/lib/docker/docker_1444137749_97536:/vagrant
    hello_world_cmx:
    hello_world_cmx: Container is starting. Output will stream in below...
    hello_world_cmx:
    hello_world_cmx: Tests Passed
```

Release

A release is a specific build deployed to a specific environment for a service. A release also includes a point in time snapshot of the service's configuration. Each versioned deployment of service on Shipped creates a unique release. Since release is a snapshot of a point in time, it cannot be modified or updated after creation.

The following table lists and describes available subcommands of release.

Release Subcommand	Description	Usage
create	Deploy a project service build to an environment.	`release create <commitId> <environmentId> <projectId> <serviceId> [optional arguments]`
get	Return one or all releases for a project and environment.	`release get <projectId> <environmentId> [<releaseId>]`
getall	Return all releases for a project and environment.	`release getall <projectId> <environmentId>`
getById	Return a specific release for a project.	`release getbyid <projectId> <releaseId>`
set	Set the default release ID.	`release set <releaseId>`

Releases [rlse] manages a release.

Command	Description	Usage	Returns

`Create`	Deploy a project service build to an environment. This command is an alias of the `shipped build deploy` command and does the exact same operations. It sets the releaseId of the new release in the CLI context for the service, so that subsequent commands (such as `rlse get`) default to fetching that release.
	Most of the optional parameters can be left to their default values, and should be modified only by users with a good understanding of Shipped and container orchestration. We recommend that new users wanting to customize optional parameters use the build deployment form provided on the Shipped web UI rather than using the CLI.
		`release create <projectId> <serviceId> <commitId> <environmentId> <optionalArguments>...`
		`<projectId> (required) The project ID of the project owing the service that was built`
		`<serviceId> (required) The service ID of the service owning the build`
		`<commitId> (required) The commit ID of the build to release`
		`<environmentId> (required) The environment ID of the environment where the build should be released`
		`ConfigID=xxx (optional) ConfigID to be deployed.`
		`ContainerCount=xxx (optional) Number of service Docker containers to deploy.`
		`ContainerCPU=xxx (optional) Number of vCPUs to allocate per container.`
		`ContainerPort=xxx (optional) Port used to access the app inside the container and that should be exposed outside the container.`
		`ContainerRAM=xxx (optional) RAM in MB to be allocated for container.`
		`DeployedURL=xxx (optional) Where the service is or would be running`
		`DeployImage=xxx (optional) The Docker image that got deployed`
		`DeploymentId=xxx (optional) Recommended default set by Shipped.`
		`EnvName=xxx (optional) Recommended default set by Shipped.`
		`EnvVariables=xxx (optional) Set additional configuration variables. These are set as env variable on Docker instances.`
			A Release struct

Command	Description	Usage	Returns
		ErrorMsg=xxx (optional) Populated if there was a problem HealthCheckMaxFailures=xxx (optional) Health check failure before marking service as down, default by buildpack. HealthCheckPath=xxx (optional) Health check endpoint URL, default by buildpack. HealthCheckProtocol=xxx (optional) Health check protocol, default by buildpack. MarathonAppID=xxx (optional) The marathon identifier for the app RepoURL=xxx (optional) The GitHub repo for the app ServiceType=xxx (optional) App, RepoApp, or Service SnapshotID=xxx (optional) If multiple services are deployed from a single deploy, they all have the same snapshotID. Useful for rollbacks	
Get	Get either all releases or a single release for a project and environment. This return basic release information status, including the endpoint URL for service access. Run with --full --json option if you need additional information such as commitId.	release get <projectId> <environmentId> [<releaseId>] <projectId> (required) ID of project for which to get releases <environmentId> (required) ID of environment for which to get releases <releaseId> (optional) The ID of the release to get; if omitted, gets all releases	Either a release struct or an array of Release structs
GetAll	Get all releases for a project and environment. This return basic release information status, including the endpoint URL for service access. Run with --full --json option if you need additional information such as commitId.	release getall <projectId> <environmentId> <projectId> (required) ID of project for which to get releases <environmentId> (required) ID of environment for which to get releases	An array of Release structs
GetById	Get a single release of a project. This return basic release information status, including the endpoint URL for service	release getbyid <projectId> <releaseId>	A release struct

Command	Description	Usage	Returns
	access. Run with `--full --json` option if you need additional information such as commitId.	`<projectId> (required) ID of project for which to get releases` `<releaseId> (required) The ID of the release to get`	
`Set`	Set the default release ID in the service's local context. This will be used as default value for subsequent API calls.	`release set <releaseId>` `<releaseId> (required) ID of the release to set as the default`	**OK** if successful

Examples

Get releases for a service:

```
shipped release get
```

```
EnvName ProjectName ServiceName Status DeployedURL   ErrorMsg
------- ----------- ----------- ------ -----------   --------
test wfproj5 wfservice5 Success http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
test wfproj5 wfservice5 Success http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
test wfproj5 wfservice5                              Error marathon app failed: wfservice5
```

Get a specific release:

```
shipped release getbyid $ e578c196-6b9b-11e5-96fe-0242ac11063c
shipped release get
```

```
EnvName ProjectName ServiceName Status   DeployedURL          ErrorMsg
------- ----------- ----------- ------   -----------          --------
test    wfproj5     wfservice5  Success  http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
```

Create release of trigger deployment of build:

```
shipped default get
```

```
name           value                                    description
----           -----                                    -----------
apiToken       cjiOgjhYeVwBTCMLfrDGXqwpzwVGqMZc
buildId        2ef3977b-6ae7-11e5-9692-0242ac11063c
commitId       16144e192f4fc3eb0ed5553f436609ed3676206a
environmentId  11b81092-6b9b-11e5-96f9-0242ac11063c
projectId      8f005490-6ada-11e5-9686-0242ac11063c     wfproj5
projectName    wfproj5
serviceId      9367df79-6ada-11e5-9687-0242ac11063c     wfservice5
serviceName    wfservice5
servicePort    38888
```

Note that sincethe above default values are set in context, the two commands below have the same effect.

```
shipped release create 8f005490-6ada-11e5-9686-0242ac11063c 9367df79-6ada-11e5-9687-0242ac11063c
16144e192f4fc3eb0ed5553f436609ed3676206a 11b81092-6b9b-11e5-96f9-0242ac11063c
shipped release create
```

```
EnvName ReleaseID                          ProjectName ServiceName Status DeployedURL ErrorMsg
------- ---------                          ----------- ----------- ------ ----------- --------
test    869139fd-6bc6-11e5-9744-0242ac11063c wfproj5     wfservice5
```

release create

Deploy a project service build to an environment. This command is an alias of the shipped build deploy command and does the exact same operations. It sets the releaseId of the new release in the CLI context for the service, so that subsequent commands (such as **rlse get**) default to fetching that release.

Usage

```
release create <commitId> <environmentId> <projectId> <serviceId> [optional arguments]
```

Exercise caution when using optional arguments. Do not change the default values unless you have a good understanding of Shipped and container orchestration. For less experience users, you can use the build deployment form provided on the Shipped Web UI to customize optional arguments.

Argument and Option

Argument and Option	Description
<commitId>	Required. The commit ID of the build to release.
<environmentId>	Required. The environment ID of the environment where the build should be released.
<projectId>	Required. The project ID of the project owing the service that was built.
<serviceId>	Required. The service ID of the service owning the build.
ConfigID=	Optional. ConfigID to be deployed.
ContainerCount=	Optional. Number of service containers to deploy.
ContainerCPU=	Optional. vCPU to allocate per container.
ContainerPort=	Optional. Port on which app runnning inside container.
ContainerRAM=	Optional. RAM in MB to be allocated for container.
DeployedURL=	Optional. Where the service is or would be running
DeployImage=	Optional. The Docker image that got deployed
DeploymentId=	Optional. Recommended default set by Shipped.
EnvName=	Optional. Recommended default set by Shipped.

Argument and Option	Description
EnvVariables=	Optional. Recommended default set by Shipped.
ErrorMsg=	Optional. Populated if there was a problem.
HealthCheckMaxFailures=	Optional. Health check failure before marking service as down, default by buildpack.
HealthCheckPath=	Optional. Health check endpoint URL, default by buildpack.
HealthCheckProtocol=	Optional. Health check protocol, default by buildpack.
MarathonAppID=	Optional. The marathon identifier for the app.
RepoURL=	Optional. The GitHub repo for the app.
ServiceType=	Optional. App, RepoApp, or Service.
SnapshotID=	Optional. If multiple services are deployed from a single deploy, they all have the same snapshotID. Useful for rollbacks

Command Example

```
shipped release create 8f005490-6ada-11e5-9686-0242ac11063c 9367df79-6ada-11e5-9687-0242ac11063c
16144e192f4fc3eb0ed5553f436609ed3676206a 11b81092-6b9b-11e5-96f9-0242ac11063c
```

Return Example

```
shipped release create

EnvName  ReleaseID                            ProjectName  ServiceName  Status  DeployedURL  ErrorMsg
-------  ---------                            -----------  -----------  ------  -----------  --------
test     869139fd-6bc6-11e5-9744-0242ac11063c  wfproj5      wfservice5
```

release get

Return one or all releases for a project and environment.

Usage

```
release get <projectId> <environmentId> [<releaseId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The project ID of the project owing the service that was built.
<environmentId>	Required. The environment ID of the environment where the build should be released.
releaseId	Optional. The release ID of the service owning the build. When no release ID is specified, all release IDs are returned.
Option	
--full	Return details
--json	Return details in JSON format.

Command Example

```
shipped release get 8f005490-6ada-11e5-9686-0242ac11063c 9367df79-6ada-11e5-9687-0242ac11063c
16144e192f4fc3eb0ed5553f436609ed3676206a
```

Return Example

```
EnvName  ProjectName  ServiceName  Status   DeployedURL                                                    ErrorMsg
-------  -----------  -----------  ------   -----------                                                    --------
test     wfproj5      wfservice5   Success  http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
test     wfproj5      wfservice5   Success  http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
test     wfproj5      wfservice5   Error                                                                   marathon app failed: wfservice5
```

release getall

Return all releases for a project and environment.

Usage

```
release getall <projectId> <environmentId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The project ID of the project owing the service that was built.
<environmentId>	Required. The environment ID of the environment where the build should be released.
Option	
--full	Return details
--json	Return details in JSON format.

Command Example

```
shipped release getall 8f005490-6ada-11e5-9686-0242ac11063c 9367df79-6ada-11e5-9687-0242ac11063c
```

Return Example

```
EnvName  ProjectName  ServiceName  Status   DeployedURL                                                    ErrorMsg
-------  -----------  -----------  ------   -----------                                                    --------
test     wfproj5      wfservice5   Success  http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
test     wfproj5      wfservice5   Success  http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
test     wfproj5      wfservice5   Error                                                                   marathon app failed: wfservice5
```

release getById

Return a specific release for a project.

Usage

```
release getbyid <projectId> <releaseId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The project ID of the project owing the service that was built.
<releaseId>	Required. The release ID of the service owning the build.

Command Example

```
shipped release getbyid 8f005490-6ada-11e5-9686-0242ac11063c 9367df79-6ada-11e5-9687-0242ac11063c
```

Return Example

```
EnvName  ProjectName  ServiceName  Status   DeployedURL                                                  ErrorMsg
-------  -----------  -----------  ------   -----------                                                  --------
test     wfproj5      wfservice5   Success  http://test--wfproj5--wfservice5--89c6a3.shipped-cisco.com
```

release set

Set the default release ID, which will be used as the default value for subsequent API calls.

Usage

```
release set <releaseId>
```

Argument and Option

Argument and Option	Description
releaseId	Required. The ID of the release to set as the default.

Command Example

```
shipped release set 16144e192f4fc3eb0ed5553f436609ed3676206a
```

Return

OK if successful.

Run

Run runs a Shipped workflow.

A Shipped workflow is an orchestrated and repeatable set of commands; that is, a set of CLI commands that run sequentially. The Run command is an alias for the `workflow run` command. See workflow for a complete explanation of Shipped workflow scripts.

Not clear what this is for

Service

A Shipped service represents a microservice within an application. Services are independently deployable and buildable. Each service has its own local and remote GitHub repository, its own CI build, and its own Docker container with everything required to build and run the service. A Shipped application can contain any number of services.

The following table lists and describes available subcommands of service.

Service Subcommand	Description	Usage
check	Verify if a service is ready.	`service check <projectId> [<serviceId>]`
create	Create a service.	
delete	Delete a service.	`service delete <projectId> <serviceId>`
get	Return a specific service.	`service get <projectId> [<serviceId>]`
getall	Return all services.	`service getall <projectId>`
repair	Repair a serivce.	`service repair <projectId> <serviceId>`
update	Update information for a service.	`service update <projectId> <serviceId> [<optional arguments>]`

service check

When Shipped creates a service, it sets up a GitHub repository and a CI build for the service. This requires coordination with external applications that occasionally are not available or have internal failures. When this occurs, the service is not ready and Shipped will not allow its project to be bootstrapped. Use the service check command to verify if a service is ready.

Usage

```
service check <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project to check for bootstrap readiness.
serviceId	Optional. The ID of service to check. If not specified, all service will be checked.

Command Example

```
shipped service check 183d5961-71f5-11e5-b058-0242ac110238
```

Return Example

```
ERROR Service hello-world-svc (5629ccba-7382-11e5-a782-0242ac11016c) is not ready: CI build could not be enabled in Drone
Service hello-world-svc2 (78e9f459-7382-11e5-a783-0242ac11016c) is ready
Service hello-world-svc3 (93ef7f44-7382-11e5-a784-0242ac11016c) is ready
ERROR Service hello-cli-svc4 (9e167ccb-7382-11e5-97c1-0242ac11045c) is not ready: GitHub repo could not be created; CI build could not be
enabled in Drone
Service hello-world-svc5 (a703d067-7382-11e5-97c2-0242ac11045c) is ready
ERROR Some services are not ready.  Use the service repair command on them before attempting bootstrap.
```

service create

You need to specify the project and buildpack when creating a service. The most convenient way to do this is the %name format supported for all UUID arguments. In addition, you can avoid specifying project ID by changing to the project context before creating a service. This is automatic if you chdir to the project directory before issuing the Shipped command.

Example

```
demo@ubuntu:~$ shipped pr cr name=hello-world desc="Sample project"

Changing to Project directory /home/demo/shipped

Changed to new project directory /home/demo/shipped/hello-world

Name         ProjectID                            Description
----         ---------                            -----------
hello-world  15dd92af-7360-11e5-a773-0242ac11016c  Sample project

demo@ubuntu:~$ cd shipped/hello-world

demo@ubuntu:~/shipped/hello-world$ shipped bp list
Name           BuildPackID
----           -----------
Dropwizard     1dd9c87b-0b3b-11e5-9cb9-7cd1c3e98f29
ExpressJS      1da3b58c-0b3b-11e5-9cb7-7cd1c3e98f29
golang         1d364ebd-0b3b-11e5-9cb3-7cd1c3e98f29
Ruby on Rails  1d6cf3c3-0b3b-11e5-9cb5-7cd1c3e98f29
Python         1d51c938-0b3b-11e5-9cb4-7cd1c3e98f29
ASP.net MVC    1dbeda73-0b3b-11e5-9cb8-7cd1c3e98f29
APICEM         0d296ec9-403a-11e5-bc38-0242ac1106ab
Grunt.JS       1d88a414-0b3b-11e5-9cb6-7cd1c3e98f29
CiscoCMX API   1df60fc4-0b3b-11e5-9cba-7cd1c3e98f29
PHP            f5e20759-f2b9-11e4-963b-14109fd75217

demo@ubuntu:~/shipped/hello-world$ shipped svc create $ %exp name=hello-world-express
Name                ServiceID                            BuildPackID                          ProjectID
----                ---------                            -----------                          ---------
hello-world-express  271c696b-7360-11e5-9240-0242ac11026c  1da3b58c-0b3b-11e5-9cb7-7cd1c3e98f29  15dd92af-7360-11e5-a773-0242ac11016c
```

service delete

Delete a service from an application.

Usage

```
service delete <projectId> <serviceId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project from which the service is deleted.
<serviceId>	Required. The ID of service to delete.

Command Example

```
shipped service delete 183d5961-71f5-11e5-b058-0242ac110238 482f255f-7389-11e5-a785-0242ac11016c
```

Return

OK if successful.

service get

Return information of one or all services for a project.

Usage

```
service get <projectId> [<serviceId>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project from which the service is returned.
serviceId	Optional. The ID of service to return. If the service ID is not specified, all services are returned.

Command Example

```
shipped service get 482f255f-7389-11e5-a785-0242ac11016c 271c696b-7360-11e5-9240-0242ac11026c
```

Return Example

```
Name                ServiceID                            BuildPackID                          ProjectID
----                ---------                            -----------                          ---------
hello-world-express 271c696b-7360-11e5-9240-0242ac11026c 1da3b58c-0b3b-11e5-9cb7-7cd1c3e98f29 15dd92af-7360-11e5-a773-0242ac11016c
```

service getall

Return a list of all services for an application. This command is particularly useful when you are in a service context and want to list all services for the project as the get command issued in a service context lists only that service.

Usage

```
service getall <projectId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project from which the service is returned.

Command Example

```
/shipped/hello-world/hello-world-cmx $ shipped svc getall
```

Return Example

```
Name                ServiceID                            BuildPackID                          ProjectID
----                ---------                            -----------                          ---------
hello-world-express 271c696b-7360-11e5-9240-0242ac11026c 1da3b58c-0b3b-11e5-9cb7-7cd1c3e98f29 15dd92af-7360-11e5-a773-0242ac11016c
hello-world-cmx     eab0ace6-7361-11e5-a775-0242ac11016c 1df60fc4-0b3b-11e5-9cba-7cd1c3e98f29 15dd92af-7360-11e5-a773-0242ac11016c
```

service repair

When Shipped creates a service, it sets up a GitHub repository and a CI build for the service. This requires coordination with external applications that occasionally are not available or have internal failures. When this occurs, the service is not ready and Shipped will not allow its project to be bootstrapped. The repair command can be used to repair a service in this state by creating the missing GitHub repository or CI build.

Usage

```
service repair <projectId> <serviceId>
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The ID of the project owning the service.
<serviceId>	Required. The ID of service to be repaired.

Command Example

```
shipped svc repair 482f255f-7389-11e5-a785-0242ac11016c 271c696b-7360-11e5-9240-0242ac11026c
```

Return

OK if successful.

service update

Update information for a service after it is created.

Usage

```
service update <projectId> <serviceId> [<optional arguments>]
```

Argument and Option

Argument and Option	Description
<projectId>	Required. The project ID of the project owing the service that was built.
<serviceId>	Required. The service ID of the service owning the build.
BuildCommand=	Optional. The shell command used to build the service.
ContainerPort=	Optional. Port on which app running inside container.
DefaultRAM=	Optional. The default RAM in MB to be allocated for the new environment for this service.
ImageSource=	Optional. The Docker image used for the service.
Name=	Optional. The new name for the service.
RunCommand=	Optional. The run command used by Docker for this service.
TestCommand=	Optional. The shell command used to run tests for the service.
EnvVariables=	Optional. Set environment variables in the service container. Use a comma-separated list when multiple variables are specified, for example, env=name1:var1,name2:var2.

Command Example

```
shipped svc upd 482f255f-7389-11e5-a785-0242ac11016c 271c696b-7360-11e5-9240-0242ac11026c DefaultRAM=512
-o "Name,ServiceID,DefaultRAM"
```

Use the **-o** option to specify the output fields, which allows the results of the update to be seen.

Return

```
Name             ServiceID                            DefaultRAM
----             ---------                            ----------
hello-world-cmx  552ab26d-7389-11e5-a786-0242ac11016c  512
```

Users

Shipped supports multiple users for application collaboration. Developers from various teams create one Shipped application and users assigned to that project can keep track of development and operation activities events in a single pane using Shipped UI.

CLI does not support adding users to application, refer to Shipped UI for adding user to the application.

The following table lists and describes available subcommands of environment.

Users Subcommand	Description	Usage
get	Get the current user or all users assigned for a specific application.	`users get [<projectId>]`
remove	Remove the current user.	`users remove <projectId>`

user get

Return information on the current user or on all users.

Usage

```
users get [<projectId>]
```

Argument and Option

Argument and Option	Description
projectId	Optional. The ID of the project for which to list the users. If the project ID is not specified, all users are returned.

Command Example

```
shipped user get 31f03bc9-0314-11e5-b9c3-6c4008ad584c
```

Return Example

```
shipped user get 31f03bc9-0314-11e5-b9c3-6c4008ad584c
Name                ID        Email
----                --        -----
John Smith          52534
Susan Suss          716385
Rick Dolan          745492
Bob White           892867
Mary Ease           1592012
```

Return Example in JSON format

```
shipped --json --full user get 31f03bc9-0314-11e5-b9c3-6c4008ad584c
[
    {
        "auth_token": "",
        "id": 52534,
        "login": "Jsmith",
        "avatar_url": "https://avatars.githubusercontent.com/u/52534?v=3",
        "html_url": "",
        "name": "John Smith",
        "email": "",
        "api_token": ""
    },
    {
        "auth_token": "",
        "id": 745492,
        "login": "rdolan",
        "avatar_url": "https://avatars.githubusercontent.com/u/745492?v=3",
        "html_url": "",
        "name": "",
        "email": "",
        "api_token": ""
    }
]
```

user remove

Remove the current user from a project.

Usage

```
users remove <projectId>
```

Argument and Option

Argument and Option	Description
projectId	Required. The ID of the project from which the current is removed.

Command Example

```
shipped user remove 31f03bc9-0314-11e5-b9c3-6c4008ad584c
```

Return Example

```
shipped user remove 31f03bc9-0314-11e5-b9c3-6c4008ad584c
Name                ID        Email
----                --        -----
John Smith          52534
```

Wizard (wz)

Wizard provides a basic tutorial to the Shipped CLI and walks you through a task workflow. Each step shows a brief explanation of its purpose and the required command to run it. You type and run the required command. Wizard workflow ends automatically when you complete the last step; however, you can terminate it any time with the quit command.

Wizard must be run in the Console mode.

The following table lists and describes available subcommands of wizard.

Wizard Subcommand	Description	Usage
curr	Review the current workflow step	`wizard curr`
wizard list.htm	List available wizard workflows.	`wizard list`
next	Go to the next step in the wizard workflow.	`wizard next`
prev	Return to the previous step in the wizard workflow.	`wizard prev`
quit	Stop and exit wizard.	`wizard quit`
start	Start a wizard.	`wizard start <workflowName>`

wizard curr

Review the current step in the wizard workflow.

Usage

```
Shipped> wizard curr
```

Example

```
List available buildpacks with the "buildpack get" command.
         bp get
Shipped> wiz curr
==> Create and Deploy step 2: List buildpacks
A buildpack is a framework base container - starter source code for a service. A Shipped service
starts from a buildpack, and you need to select a buildpack when creating a service.

List available buildpacks with the "buildpack get" command.
         bp get
Shipped>
```

wizard list

Return a list of available wizard workflows.

Usage

```
Shipped> wizard list
```

Example

```
Shipped> wiz list
    Workflow Name      Description
    -------------      -----------
1.  Create and Deploy  Create a project and service and deploy to an environment
Select wizard to start or press Enter to continue:
```

wizard next

Go to the next step in the wizard workflow.

You can also press Enter with no input after a successful command to advance to the next step.

Usage

```
Shipped> wizard next
```

Example

```
The first step is to create a project with the "project create" command.
         project create name=xxx [desc="description"]
Shipped> project create name=test
Changed to new project directory c:\work\shipped\test
Name  ProjectID                            Description
----  ---------                            -----------
test  d98b59fb-6cec-11e5-97ed-0242ac11063c
Press Enter with no input to show the next step in wizard workflow Create and Deploy
Shipped:test> wiz next
==> Create and Deploy step 2: List buildpacks
A buildpack is a framework base container - starter source code for a service. A Shipped service
starts from a buildpack, and you need to select a buildpack when creating a service.

List available buildpacks with the "buildpack get" command.
              bp get
Shipped:test>
```

wizard prev

Return to the previous step in the wizard workflow.

Usage

```
Shipped> wizard prev
```

Example

```
List available buildpacks with the "buildpack get" command.
         bp get
Shipped> wiz prev
==> Create and Deploy step 1: Create project
A Shipped project is a single entity that holds a collection of services. Each service is stored in
its own GitHub repository and can be edited and deployed independently In this workflow, we create
a project, add a service, create local and remote repositories, and deploy the completed project to
the cloud

The first step is to create a project with the "project create" command.
         project create name=xxx [desc="description"]
Shipped>
```

wizard quit

Wizard workflow ends automatically when you complete the last step. Use this command to stop and exit wizard anytime.

Usage

```
Shipped> wizard quit
```

Example

```
List available buildpacks with the "buildpack get" command.
         bp get
Shipped> wiz q
OK
Shipped>
```

wizard start

Start a wizard.

Usage

```
Shipped> wizard start <workflowName>
```

Argument and Option

Argument and Option	Description
<workflowName>	Required. The name (can be abbreviated) of the workflow to run.

Example

```
Shipped> wiz start create
==> Create and Deploy step 1: Create project
A Shipped project is a single entity that holds a collection of services. Each service is stored in
its own GitHub repository and can be edited and deployed independently In this workflow, we create
a project, add a service, create local and remote repositories, and deploy the completed project to
the cloud

The first step is to create a project with the "project create" command.
         project create name=xxx [desc="description"]
Shipped>
```

Workflow (wf)

A workflow carries out an orchestrated and repeatable set of commands that performs a high-level operation, such as creating a project with a service and deploying it to the cloud. Rather than manually executing a number of command, you start a workflow with the Shipped **run** command, and Shipped executes the commands in sequence until it either reaches the end of the workflow, or a command fails.

You can supply arguments to a workflow on the **run** command itself, or allow the workflow to prompt for argument values.

You can use Cisco workflow commands, or create your own workflow.

Need XRF -- creating own wf

The following table lists and describes available subcommands of workflow.

Workflow Subcommand	Description	Usage
help	Show help information for a workflow.	`wf help <workflowName>`
list	Return a list of workflows.	`wf list [workflowName]`
resume	Continue execution of a workflow from the point of its failure.	`resume [--arg1 value1 [--arg2 value2...]] [--help] [--verbose]`
run	Run a Shipped workflow.	`run workflowName [--arg1 value1 [--arg2 value2...]] [--help] [--verbose]`

workflow help

Show help information for a workflow.

Usage

```
wf help <workflowName>
```

Argument and Option

Argument and Option	Description
workflowName	Required. The name of the workflow. There are two predefined workflow commands currently available: • **create-and-deploy** - Create, bootstrap, and deploy a Shipped project with a single service. • **fast-deploy** - Fast bootstrap and deploy a Shipped project by name.

Command Example

```
wf help create-and-deploy
```

Return Example

```
Workflow create-and-deploy
  Create, bootstrap, and deploy a Shipped project with one service

  Usage:
  run create-and-deploy --project xxx --service xxx --framework xxx [--env xxx] [fast=true]

  Shipped prompts for any required variable not provided in the command line.

  Workflow arguments:
  project (prompted): the name of the project
  service (prompted): the name of the service
  framework (prompted): the name of the service framework
  env (optional, default "staging"): the name of the deployment environment
  fast (optional, default ""): specify fast=true for fast bootstrap
```

workflow list

Return a list of workflows.

Usage

```
wf list [workflowName]
```

Argument and Option

Argument and Option	Description
workflowName	Optional. The name of the workflow. If no workflowName is specified, or if matches more than one workflow, the command returns the names, source, and description of all available workflow.

Command Example

```
wf list
```

Return Example

Source can be one of three values:

- **local** – the workflow is stored on your local laptop in directory $HOME/.shipped/scripts.
- **localCopy** – the workflow is stored on your local laptop and has the same name as a remote workflow. Shipped uses the local copy in preference to the remote version.
- **remote** – the workflow comes from a cloud server.

```
Name              Source  Description
----              ------  -----------
create-and-deploy  remote  Create, bootstrap, and deploy a Shipped project with one service
fast-deploy        remote  Fast bootstrap and deploy a Shipped project by name
```

workflow resume

In console mode, continue execution of a workflow from the point of its failure.

Question: Does workflow resume work only in console mode?

Usage

```
resume [--arg1 value1 [--arg2 value2...]] [--help] [--verbose]
```

Argument and Option

Argument and Option	Description
--arg value	Optional. arguments with values.
--help	Optional. Show help text for the workflow.
--verbose	Optional. List each workflow command before execution.

Command Example

```

```

Return Example

Question: Need command and Return command examples

workflow run

Run a Shipped workflow.

Usage

```
run workflowName [--arg1 value1 [--arg2 value2...]] [--help] [--verbose]
```

Argument and Option

Need info

Argument and Option	Description
workflowName	Required. The name of the workflow to run.
--arg value	Optional. Arguments with values, or you can use key=value pair in the alternate syntax argn=valuen. To list arguments for a workflow, run wf help workflowName.
--help	Optional. Show help text for the workflow.
--verbose	Optional. List each workflow command before execution.

Command Example

```
wf run create-and-deploy --project hello-world --service cmx-service --framework cmx
```

Return Example

```
Here are the names of the available service frameworks
Name
----
Dropwizard
ExpressJS
golang
Ruby on Rails
Python
ASP.net MVC
APICEM
Grunt.JS
CiscoCMX API
PHP

Creating project hello-world
Changed to new project directory c:\work\shipped\hello-world
Name         ProjectID                            Description
----         ---------                            -----------
hello-world  8a137b52-6c71-11e5-97d5-0242ac11063c

Creating service cmx-service using framework cmx
Name         ServiceID                            BuildPackID                          ProjectID
----         ---------                            -----------                          ---------
cmx-service  8debe5b1-6c71-11e5-97d6-0242ac11063c  1df60fc4-0b3b-11e5-9cba-7cd1c3e98f29  8a137b52-6c71-11e5-97d5-0242ac11063c

Bootstrapping project hello-world
git 2.5.3 is installed.
============================================================
Cloning Repositories
============================================================
Cloning service 1 cmx-service (8debe5b1-6c71-11e5-97d6-0242ac11063c) into hello-world/cmx_service
Cloning into 'hello-world/cmx_service'...
...
```

Ellipsis (…) indicates more output follows, but not display for brevity purpose.