

Cisco Intercloud Services



Shipped CLI Guide

Published 12/14/2015



Copyright Notice

Copyright © 2014-2015 Cisco Systems, Inc. and/or Its affiliates. All Rights Reserved.

The information contained in this document is proprietary and confidential to Cisco Systems, Inc. ("Cisco") and/or its affiliates, and is furnished in confidence to you under the Confidentiality terms of the applicable agreement between you and Cisco, with the understanding that it will not, without the express written permission of Cisco, be used or disclosed for other than for the purposes set forth in that agreement.

Information about Cisco Services and technology may be subject to export controls under the laws of the United States and other countries. You and Cisco shall comply with such laws and you agree not to knowingly export, re-export or transfer such information without first obtaining any required United States or any other applicable authorizes or licenses.

The trademarks, logos and service marks ("Marks") displayed in this document are the property of Cisco or third parties. Users are not permitted to use these Marks without the prior written consent of Cisco or such third party which may own the Mark. "Cisco" is a registered trademark of Cisco and/or its affiliates.

The design in this document may contain or reference software from the open source community, including OpenStack® technology, that must be licensed under the specific license terms applicable to such software.

Document History

Topic	Date of Change	Description
Initial publication	12/8/15	

About This Document

This document describes the Shipped CLI commands.





Audience

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

Related Documents

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

Conventions

Item	Description
bold	Menu, command.
mono-space font	Code, typed data.
<i>italic</i> 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:

<http://intercloud.cisco.com/support>

Contents

CLI Guide	1
Copyright Notice	2
Document History	3
About This Document	4
Audience	4
Related Documents	4
Conventions	4
Support	4
Shipped CLI	7
Help	7
Usage	7
Authentication	9
Command Structure	9
Default Values and Context Sensitivity	9
The Dollar Sign Placeholder	10
Shipped Console Mode	11
BuildPack	11
Builds	12
Config	16
Console	18
Defaults	18
Environments	19
Local	21
Project	24
Releases	27
Run	30
Services	31
Users	34
Wizard	36
Workflow	39

Glossary	42
----------------	----

Shipped CLI

The Shipped CLI provides command line access to Shipped services. This includes services that can be performed with the Shipped web 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.

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**.

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
<code>shipped help</code>	Overview of the CLI (same information as this page)
<code>shipped help <i>command</i></code>	Help on command <i>command</i>
<code>shipped help <i>command subcommand</i></code>	Help on <i>subcommand</i> of <i>command</i>
<code>shipped help workflow</code>	Overview of Shipped orchestrated workflow
<code>shipped wf help <i>workflowName</i></code>	Help on individual workflow script <i>workflowName</i>
<code>shipped help default</code>	Context sensitivity and default values features

Usage

```
shipped [options] [command [subcommand [args]]]
```

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.

Commands

Click a command to see more information.

Command	Alias
BuildPack	bp
Builds	bld
Config	cfg
Defaults	dflt
Environments	env
Local	lcl
Project	pr
Releases	rlse

Command	Alias
Run	
Services	svc
Users	usr
Wizard	wz
Workflow	wf

Subcommands and Arguments

Subcommands and arguments vary for each command.

Authentication

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 Structure

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.

You can abbreviate commands, subcommands, and arguments by truncation to the minimum length necessary for unambiguous specification. You can also use aliases for a command.

Default Values and Context Sensitivity

Many commands use one or more positional arguments that are usually UUIDs. For example, the service get syntax is:

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

This shows that the command has two arguments: a required argument of a project ID and an optional argument of a service ID. To supply these arguments:

- **Invoke the CLI from a project or service directory.** This causes the project ID argument to default to the ID of the project. If you invoke the CLI from a service directory, the service ID also defaults. The Shipped CLI is context sensitive. You can default multiple project and service IDs. See [Defaults](#) for more information.
- **Supply the first few characters of the UUID Docker-style.** For example, if the project ID was "c272e20d-6ab4-11e5-967e-0242ac11063c", you could specify the command `svc get c27`. You only need enough leading characters of the UUID to uniquely identify the project. You can specify UUIDs in this fashion for builds, buildpacks, environments, projects, and services.
- **Supply a portion of the name of the project or service, preceded by a percent sign.** For example, if the project was named "Hello-World" and the service was named "my-php-service", you could specify the command `svc get %wor %php`. Name specifications are not case sensitive and can be any part of the name (not necessarily the beginning). Only enough characters to uniquely identify the project or service are needed. You can specify UUIDs in this fashion for builds, buildpacks, environments, projects, and services.

The Dollar Sign Placeholder

Use a single dollar sign (\$) in the argument position to use the default value.

For example, to use `svc get` to show the CMX service for the current project, use the command:

```
shipped svc get $ %cmx
```

The dollar sign placeholder uses the default project ID, while the `%cmx` argument uses the UUID of the service with "CMX" somewhere in its name. You only need to specify the service explicitly like this if you were not in the context of the service (for example, you were issuing the command from the project directory or from the directory of some other service).

Shipped Console Mode

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 the command:

```
shipped console
```

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

Once in console mode:

- Don't use the "shipped" prefix on commands.
- Scroll through a history of previous commands using the up and down arrow keys.
- Press the tab key to complete command names and arguments.
- Use the `resume` or `wf resume` to resume execution of a failed workflow at the line 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.

BuildPack

BuildPack [bp] is the starter source code for developing a service.

Command	Description	Usage	Returns
Get	Get one or all buildpacks.	<code>shipped buildpack get [<buildpackId>]</code> <buildpackId> - (optional) ID of the buildpack to get; if omitted, returns all buildpacks	BuildPack struct or an array of buildpack structs
GetAll	Get a list of available buildpacks	<code>shipped buildpack getall</code>	An array of buildpack structs
Set	Sets the default buildpack ID.	<code>shipped buildpack set <buildpackId></code> <buildpackId> (required) ID of the buildpack to set as the default	OK if successful

Examples

For complete details of buildpack fields, use option `--full`.

To view in JSON, use `--json` while fetching the buildpack list.

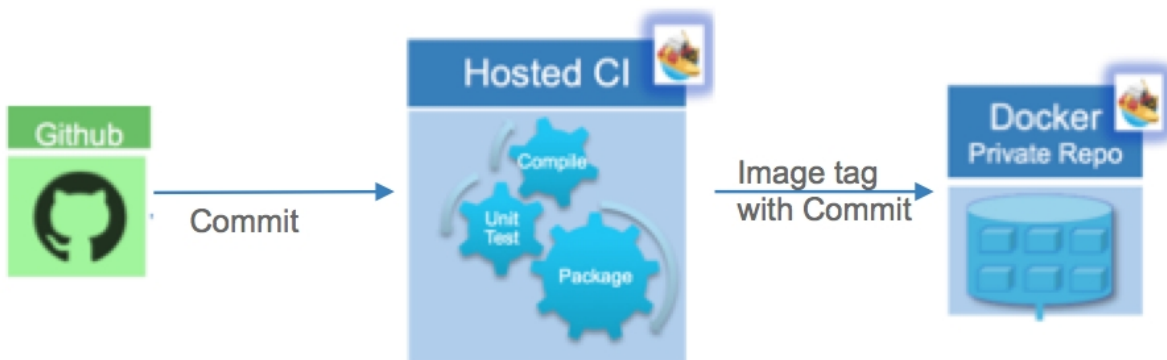
```
shipped --json --full buildpack get %express
```

To set a default buildpack for your project, use the `set` command option. A successful set is confirmed by an OK response.

```
shipped buildpack set %python
```

Builds

Builds [bld] creates status and logs for a service.



Command	Description	Usage	Returns
Get	Get build details for a single service or for all services in a project	<pre>build get <projectId> [<serviceId>] [--latest] projectId - (required) ID of project for which to get builds serviceId - (optional) ID of service for which to get build; if omitted, gets builds for all services</pre>	An array of build structs

Command	Description	Usage	Returns
		--latest - (optional) Retrieve only the latest (most recent) build	
GetLogs	Get the logs for a specified 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.	build getlogs <projectId> <serviceId> <buildId> projectId - (required) ID of project with service for which build logs are desired serviceId - (required) ID of service for which build logs are desired buildId - (required) ID of build for which logs are desired	The logs for one service build in a project
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.	build set <projectId> <serviceId> [<buildId>] [--latest] <buildId> - (required) build ID to set as the default <projectId> - (required) ID of the project owning the build <serviceId> - (required) ID of the service built by the build --latest - (optional) Set build ID to that of the most recent build (and ignore the <buildId> argument)	OK if successful
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. 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.	build deploy <projectId> <serviceId> <commitId> <environmentId> optionalArguments... <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 owning the service that was built <serviceId> (required) The service ID of the service owning the build	Release struct

Command	Description	Usage	Returns
		<p>ConfigID=xxx (optional) ConfigID to be deployed.</p> <p>ContainerCount=xxx (optional) Number of service Docker containers to deploy.</p> <p>ContainerCPU=xxx (optional) Number of vCPUs to allocate per container.</p> <p>ContainerPort=xxx (optional) Port used to access the app inside the container and that should be exposed outside the container.</p> <p>RAM=xxx (optional) RAM in MB to be allocated for container.</p> <p>DeployedURL=xxx (optional) Where the service is or would be running</p> <p>DeployImage=xxx (optional) The Docker deployed image</p> <p>DeploymentId=xxx (optional) Recommended default set by Shipped.</p> <p>EnvVariables=xxx (optional) Set additional configuration variables. These are set as env variable on Docker instances.</p> <p>ErrorMsg=xxx (optional) Populated if there was a problem.</p> <p>HealthCheckMaxFailures=xxx (optional) Health check failure before marking service as down, default by buildpack.</p> <p>HealthCheckPath=xxx (optional) Health check endpoint URL, default by buildpack.</p> <p>HealthCheckProtocol=xxx (optional) Health check protocol, default by buildpack.</p> <p>MarathonAppID=xxx (optional) The marathon identifier for the app</p> <p>RepoURL=xxx (optional) The GitHub repo for the app</p>	

Command	Description	Usage	Returns
		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 rollback	

Examples

Get a build list from outside a bootstrapped local development environment:

```
shipped build get %projectName %serviceName --latest
```

Since this command is executed outside the development environment, `projectid` and `serviceid` must be passed explicitly to `build get` either as a name preceded by a per cent sign or as a full or partial UUID. If you go to the `projectname/sevicename` directory in the bootstrapped environment, then the Shipped CLI auto-detects the project and service IDs, and these parameters can be skipped while making the call.

```
cd demoproject/demoservice
shipped build get
```

Fetch logs for a specific build:

```
shipped build getlogs $ $ 2ef
```

In this example, 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.

Set a specific `buildid` and `commitid` as defaults for subsequent calls to the Shipped CLI:

```
shipped build set $ $ 2ef
```

This command sets defaults for build ID and commit ID, allowing them to be omitted in future commands.

Deploy a specific build to an environment after using the above set command:

```
shipped build deploy
```

This command has been issued from the service directory after build and commit ID defaults have been set, so positional arguments can be omitted.

Config

Config [cfg] enables you to show and modify config files for your project

Command	Description	Usage	Returns
Delete	Delete an environment configuration	<pre>config delete <projectId> <serviceId> <environmentId> <configId> <projectId> - (required) ID of project with config to delete <serviceId> - (required) ID of service with config to delete <environmentId> - (required) ID of project environment with config to delete <configId> - (required) ID of config to delete</pre>	OK if successful
GetAll	Get all configurations	<pre>config getall <projectId> <serviceId> <environmentId> <projectId> - (required) ID of project <serviceId> - (required) ID of one project's services <environmentId> - (required) ID of a project's service environment</pre>	An array of ServiceConfig structs
Set	Set the default config ID	<pre>config set <configId> <configId> - (required) ID of the config to set as the default</pre>	OK if successful
Update	Update a project configuration	<pre>config update <projectId> <serviceId> <environmentId> <configId> <optionalArgs>...</pre>	

Command	Description	Usage	Returns
		<p><projectId> - (required) ID of project with config to update</p> <p><serviceId> - (required) ID of service with config to update</p> <p><environmentId> - (required) ID of project environment with config to update</p> <p><configId> - (required) ID of config to update</p> <p>ContainerCount=xxx - (optional) The actual number of containers Marathon will deploy in this environment</p> <p>ContainerCPU=xxx - (optional) The actual CPU allocation marathon will use for deployments to this environment</p> <p>ContainerPort=xxx - (optional) The actual port marathon will use for deployments to this environment</p> <p>ContainerRAM=xxx - (optional) The actual RAM allocation marathon will use for deployments to this environment</p> <p>Deployimage=xxx - (optional) The image (usually found at registry.hub.docker.com) of the Docker container in which to run</p>	

Examples

Get all configurations (config getall):

Issuing the command in the context of a service defaults projectId, serviceId, and envId.

```
demo@ubuntu:~/shipped/test-proj/test-svc: shipped cfg getall
```

Set default configuration (config set):

Issuing the command in the context of a service defaults projectId, serviceId, and envId.

```
demo@ubuntu:~/shipped/test-proj/test-svc: configs set b1d59858-6d57-11e5-aa3e-0242ac115798
```

Update a configuration (config update):

Issuing the command in the context of a service defaults projectId, serviceId, and envId.

```
demo@ubuntu:~/shipped/test-proj/test-svc: cfg upd containerram=128 containerport=3000 containercount=1  
containercpu=0.5 deployimage="mikejihbe/nodejs-ssh:latest"
```

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 the command:

```
shipped console
```

To exit console mode, enter the command `exit` or `quit` (abbreviated to `q`).

The following features are available in console mode:

- You don't need the "shipped" prefix on commands (and must not use it in console mode).
- Use the up and down arrows to scroll through a history of previous commands.
- Press the tab key to complete command names and arguments.
- Use the `resume` (or `wf resume`) command to continue execution of a failed workflow at the line of the failure.
- You can use Docker-style or `%xxx` references to extract UUIDs from the preceding command's output. This allows you to conveniently supply ID values for objects other than builds, buildpacks, environments, projects, and services.

Defaults

Defaults [`dflt`] enables you to show and set default values.

To get the current set of defaults, use:

```
shipped dflt get
```

You can add variables to the list of defaults with the set subcommand supported by most commands. For example, you can add build and commit IDs to the list of defaults with the command:

```
shipped build set <projectId>
```

This command finds the build ID and commit ID from the most recent build for the project and adds them to the list of defaults.

Command	Description	Usage	Returns
get	Get 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.	defaults get	A list of default values

Example

Show defaults from a service directory:

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

name	value	description
----	-----	-----
apiToken	RLUtXXXXXXXXXXXXXXXXXXXXX	Iub
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	

Environments

Environments [env] manages namespace deployment.

Command	Description	Usage	Returns
Create	Create a project environment. Create environment internally triggers creation of a configuration for every service in the project with a default deploy target.	environment create <projectId> name=xxx [description=xxx] <projectId> (required) The project ID for the new environment Name=xxx (required) The name of the environment Description=xxx (optional) A description of the environment	An environment struct
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 complete deployment environment is needed.	environment delete <projectId> <environmentId> <projectId> (required) The project ID for which to delete an environment <environmentId> (required) The ID of the environment to delete	OK if successful
Get	Get one or all project environments.	environment get <projectId> [<environmentId>] <projectId> (required) The project ID of the environment(s) <environmentId> (optional) The environment ID; if omitted, all environments	Either an Environment struct or an array of Environment structs
GetAll	Get all environments.	environment getall <projectId> <projectId> (required) The project ID of the environments	An array of Environment structs
Set	Set the default environment ID		

Examples

To create a new environment for a project:

```
cd <projectName>
shipped env create name=test description="Enviroment to deploy test cluster"
```

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

Issuing the command from the project directory allows the projectId to be omitted from command arguments.

To set the environment ID as the default:

```
shipped env set 11b8
```

OK

```
shipped default get
```

name	value	description
environmentId	11b81092-6b9b-11e5-96f9-0242ac11063c	
projectId	8f005490-6ada-11e5-9686-0242ac11063c	wfproj5
serviceId	9367df79-6ada-11e5-9687-0242ac11063c	wfservice5

The Environment ID used during build deploy. This example uses the default envId set above, and default values for other params, such as projectId and buildId.

```
shipped build deploy
```

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

Local

Local [lcl] 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'll need to run bootstrap again before using most of the local commands.

The VM uses VirtualBox and is managed by Vagrant, both open source applications. Shipped bootstrap installs them for you if necessary.

Command	Description	Usage	Returns
bootstrap	Bootstrap a Shipped project. Bootstrapping a project sets it up on your local laptop by: <ul style="list-style-type: none"> Setting up a local Git repository for each of the project's services and cloning the remote GitHub repository set up by Shipped. If necessary, installing prerequisite software (VirtualBox and Vagrant) Creating and starting a VM with a Docker container for each of the project's services 	<pre>lcl bootstrap <projectId> [-- fast --localDocker]</pre>	--
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.	<pre>lcl build <projectId> <serviceId></pre>	Output from the build
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	<pre>lcl commit <projectId> [<serviceId>] message=xxx</pre>	Git output

Command	Description	Usage	Returns
	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.		
destroyvm	Destroys the VM running project services. This command removes all files associated with the VM from your laptop. After it completes, you will not be able to build and test services locally until you recreate the VM with the local bootstrap command	lcl destroyvm <projectId> [<serviceId>]	--
haltvm	Halts the VM running project services. This command stops the VM, but leaves its files intact so that it can be restarted quickly. After it completes, you will not be able to build and test services locally until you restart the VM with either the local reloadVM command or thelocal bootstrap command.	lcl haltvm <projectId> [<serviceId>]	--
logs	Show log files that a service has written to its container.	lcl logs <projectId> [<serviceId>]	The content of the project's logs
reloadvm	Halts and restarts the VM running project services. This command recycles the VM, including restarting any web server resident on the VM.	lcl reloadvm <projectId> [<serviceId>]	--
statusvm	Shows 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.	lcl statusvm <projectId> [<serviceId>]	--
test	Test a service in its container. 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.	lcl test <projectId> <serviceId>	Output from the service's unit tests

Examples

To bootstrap the local environment:

```
demo@ubuntu:~/shipped: shipped local bootstrap
```

To build a service on its VM:

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

To commit changes to a project:

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl commit m="First commit"
```

To destroy the VM running project services:

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl destroy
```

To stop the VM running project services:

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

To view the log files:

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl logs
```

To stop and restart the VM running project services:

```
demo@ubuntu:~/shipped/hello-world/hello_world_cmx$ shipped lcl reloadvm
```

To see the project VM status and its service containers:

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

To test a service in its container:

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

Project

Project [pr] manages a Shipped project.

Command	Description	Usage	Returns
Create	Create a project.	project create Name=xxx [Description=xxx]	A Project struct

Command	Description	Usage	Returns
		Name - (required) name of the new project Description - (optional) description of the project	
Delete	Delete a project and associated data, including services and deployment environments. Shipped confirms you really want to do this before proceeding.	project delete <projectId> <projectId> - (required) ID of project to delete	OK if successful
Get	Retrieve details for one or all projects.	project get [<projectId>] <projectId> - (optional) ID of project to get; if omitted, returns all projects	Either a Project struct or an array of Project structs
GetAll	Retrieve a list of all projects. This command is useful when you are in a project context and want to list all projects, as the get command issued in a project context lists only that project.	project getall	An array of Project structs
Update	Update a project.	project update <projectId> name=xxx [description=xxx] <projectId> - (required) ID of project to update Description - (optional) updated description of the project	A Project struct

Examples

To list a single project:

Use the get subcommand to show information about a single project. You can specify the project by ID, partial ID (Docker style), or by a name or partial name preceded by a percent sign.

For example, if you have a project named "godemo" with ID 183d5961-71f5-11e5-b058-0242ac110238, all of these commands list it:

```
shipped project get 183d5961-71f5-11e5-b058-0242ac110238
shipped project get 183
shipped pr get %go
shipped pr get %demo
```

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

To list all projects:

Use the get subcommand without an argument to list all projects. You can also use the getall subcommand. Getall is useful if you're in a project context (for example, invoking the CLI from a project directory), as in this case the default functionality of get is to list that project.

```
shipped get
shipped getall
```

Name	ProjectID	Description
clumsy	93407de2-679f-11e5-8425-0242ac113ce8	
godemo-new	183d5961-71f5-11e5-b058-0242ac110238	New Go project
kanboard	7e7e2b43-6861-11e5-851d-0242ac113ce8	

To create a project:

```
shipped project create name="TestProject" Description="A sample project"
shipped pr cr n="TestProject" d="A sample project"
```

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

To update a project:

```
demo@ubuntu:~ $ shipped project upd %api desc="Demo Shipped API"
```

id	name	description	last_seen_event_id	event_count	user_count
f10e6237-72bf-11e5-9202-0242ac11026c	demo-api-issue	Demo Shipped API		0	0

To delete a project:

```
demo@ubuntu:~ $ shipped pr del %demo
```

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

Releases

Releases [rlse] manages a release.

Command	Description	Usage	Returns
Create	<p>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 releaseld of the new release in the CLI context for the service, so that subsequent commands (such as rlse get) default to fetching that release.</p> <p>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.</p>	<pre>release create <projectId> <serviceId> <commitId> <environmentId> <optionalArguments>...</pre> <p><projectId> (required) The project ID of the project owning the service that was built</p> <p><serviceId> (required) The service ID of the service owning the build</p> <p><commitId> (required) The commit ID of the build to release</p> <p><environmentId> (required) The environment ID of the environment where the build should be released</p> <p>ConfigID=xxx (optional) ConfigID to be deployed.</p> <p>ContainerCount=xxx (optional) Number of service Docker containers to deploy.</p> <p>ContainerCPU=xxx (optional) Number of vCPUs to allocate per container.</p> <p>ContainerPort=xxx (optional) Port used to access the app inside the container and that should be exposed outside the container.</p>	A Release struct

Command	Description	Usage	Returns
		<p>ContainerRAM=xxx (optional) RAM in MB to be allocated for container.</p> <p>DeployedURL=xxx (optional) Where the service is or would be running</p> <p>DeployImage=xxx (optional) The Docker image that got deployed</p> <p>DeploymentId=xxx (optional) Recommended default set by Shipped.</p> <p>EnvName=xxx (optional) Recommended default set by Shipped.</p> <p>EnvVariables=xxx (optional) Set additional configuration variables. These are set as env variable on Docker instances.</p> <p>ErrorMsg=xxx (optional) Populated if there was a problem</p> <p>HealthCheckMaxFailures=xxx (optional) Health check failure before marking service as down, default by buildpack.</p> <p>HealthCheckPath=xxx (optional) Health check endpoint URL, default by buildpack.</p> <p>HealthCheckProtocol=xxx (optional) Health check protocol, default by buildpack.</p> <p>MarathonAppID=xxx (optional) The marathon identifier for the app</p> <p>RepoURL=xxx (optional) The GitHub repo for the app</p> <p>ServiceType=xxx (optional) App, RepoApp, or Service</p> <p>SnapshotID=xxx (optional) If multiple services are deployed from a single deploy, they all have the same snapshotID. Useful for rollbacks</p>	
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 -	<pre>release get <projectId> <environmentId> [<releaseId>]</pre>	Either a release struct or

Command	Description	Usage	Returns
	-full --json option if you need additional information such as commitId.	<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	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 access. Run with --full --json option if you need additional information such as commitId.	release getbyid <projectId> <releaseId> <projectId> (required) ID of project for which to get releases <releaseId> (required) The ID of the release to get	A release struct
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 since the 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			

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.

Services

Services [svc] manages project services. Services are independently deployable components of a project.

Command	Description	Usage	Returns
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 this command to verify that a service is ready. If it's not, use service repair.	<pre>service check <projectId> [<serviceId>] <projectId> - (required) the project ID of the project to check for bootstrap readiness <serviceId> - (optional) the ID of the service to check; if omitted, checks all services</pre>	OK if successful
Create	Creates a service for a project.	<pre>service create <buildPackId=xxx DeployImage=xxx> name=xxx ... <projectId> - (required) the project ID of the project for which to create the service <buildPackId> - (optional) - The id of the buildpack to use for the service. If buildPackId is omitted, DeployImage must be provided. Name=xxx (required) - The name of the service Optional Arguments BootstrapTemplateRepo=xxx - The GitHub repository that hosts a sample hello-world implementation to use for a new service. BuildCommand=xxx - The shell command to build the service BuildImage=xxx - Build image used when building a service to be tested with Drone.</pre>	A Service struct

Command	Description	Usage	Returns
		<p>ContainerPort=xxx - The container port used by Docker for this service</p> <p>ContainerSharedDirectory=xxx - The location within the Docker container that will be shared with the host.</p> <p>CPU=xxx - The number of CPUs required for the service</p> <p>Deployimage=xxx - The image (usually found at registry.hub.docker.com) of the Docker container in which to run</p> <p>EnvVariables=x:x (optional) One or more environment variables to set in the service's container. Specify as a comma-separated list in the form env="name1:var1,name2:var2"</p> <p>ImageSource=xxx - A URL for an image used by this service.</p> <p>Organization=xxx - Public=xxx - Whether or not repository should be public (default true)</p> <p>RAM=xxx - The amount of RAM required for the service</p> <p>Repository=xxx - The GitHub repository for the service in the form https://github.com/username/reponame (will be created if it doesn't already exist)</p> <p>RunCommand=xxx - The shell command to run the service</p> <p>TestCommand=xxx - The shell command to run tests for the service</p>	
Delete	Deletes a service from a project.	<pre>service delete <projectId> <serviceId> <projectId> - (required) the project ID for the service to delete <serviceId> - (required) the ID of the service to delete</pre>	OK if successful
Get	Gets information about a single service or about all services for a project.	<pre>service get <projectId> [<serviceId>] <projectId> - (required) the project ID for which to list service(s) <serviceId> - (optional) the ID of the service to list; if omitted, lists all services</pre>	Either a Service struct or an array of Service structs
GetAll	Retrieve a list of all services for a project. This command is particularly useful when you are in a service context and want to list all services for the project, as the get	<pre>service getall <projectId></pre>	An array of Service structs

Command	Description	Usage	Returns
	command issued in a service context lists only that service.		
Repair	Repairs a service that can't be bootstrapped.	<pre>service repair <projectId> <serviceId> <projectId> - (required) the project ID for the project owning the service <serviceId> - (required) the service ID of the service to be repair</pre>	OK if successful
Update	Updates information for a service. You can change the name or other information about a service after creating it.	<pre>service update <projectId> <serviceId> <optionalArguments...> <projectId> (required) - The project ID of the project owning the service <buildpackId> (required) - The id of the buildpack to use for the service Optional Arguments: BuildCommand=xxx - The shell command to build the service ContainerPort=xxx - The container port used by Docker for this service DefaultRAM=xxx - The default RAM allocation for new environments for this service in Marathon EnvVariables=x:x (optional) One or more environment variables to set in the service's container. Specify as a comma-separated list in the form env="name1:var1,name2:var2" ImageSource=xxx - The Docker image to use for the service Name=xxx - A new name for the service RunCommand=xxx - The run command used by Docker for this service TestCommand=xxx - The shell command to run tests for the service</pre>	A Service struct

Examples

To create a service:

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.

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

To see information about a service:

You can use the get subcommand to show information about a single service. You can specify the service by id, partial ID (Docker style), or by a name or partial name preceded with a per cent sign.

For example, if your project "hello-world" has a service named "hello-world-express" with ID 271c696b-7360-11e5-9240-0242ac11026c, all of these commands list it:

```
demo@ubuntu:~$ shipped service get %hello %hello
demo@ubuntu:~/shipped/hello-world$ shipped svc get $ 271
demo@ubuntu:~/shipped/hello-world$ shipped svc get $ 271c696b-7360-11e5-9240-0242ac11026c
```

To list all services in a project:

```
demo@ubuntu:~/shipped/hello-world/hello-world-cmx $ shipped svc getall
```

To check a service for bootstrapping:

```
demo@ubuntu:~/shipped/hello-world $ shipped svc check
```

To modify a service:

```
demo@ubuntu:~/shipped/hello-world/hello-world-cmx $ shipped svc get -o "Name,ServiceID,DefaultRAM"
```

To delete a service:

```
demo@ubuntu:~/shipped/hello-world $ shipped svc getall
```

Users

Users [usr] manages project users.

Command	Description	Usage	Returns
Get	Get the current user or all users assigned for a specified project.	<code>users get [<projectId>]</code>	Either a user struct or an

Command	Description	Usage	Returns
		<projectId> (optional) ID of the project for which to get users; if omitted, gets the current user	array of user structs
Remove	Removes the current user from a project.	users remove <projectId> <projectId> (required) ID of the project for which to remove the current user	The user struct of the removed user

Examples

Get all users provisioned for specific project by id:

Email id is displayed if the email is set as public viewable in the user's GitHub profile.

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

Name	ID	Email
----	--	-----
Mike Ihbe	52534	
Brendan McElligott	716385	
Ian Kinsey	745492	
Matt Nish	892867	
Robby Polana	1592012	
Brad Gearon	1731943	
Neelesh Pateriya	5516389	

Listing users with all fields in JSON format:

```
ShippedDemo>>shipped --json --full user get 31f03bc9-0314-11e5-b9c3-6c4008ad584c
```

```
[
{
  "auth_token": "",
  "id": 52534,
  "login": "mikejihbe",
  "avatar_url": "https://avatars.githubusercontent.com/u/52534?v=3",
  "html_url": "",
  "name": "Mike Ihbe",
  "email": "",
  "api_token": ""
},
{
  "auth_token": "",
  "id": 716385,
  "login": "mcelligott",
  "avatar_url": "https://avatars.githubusercontent.com/u/716385?v=3",
  "html_url": "",
  "name": "Brendan McElligott",
  "email": "",
  "api_token": ""
},
{
  "auth_token": "",
  "id": 5516389,
  "login": "npateriya",
  "avatar_url": "https://avatars.githubusercontent.com/u/5516389?v=3",
  "html_url": "",
  "name": "",
  "email": "",
  "api_token": ""
}
]
```

Wizard

Wizard [wz] is the interactive execution of a Shipped workflow.

Wizards provide an introduction to the Shipped CLI and walk you through a task workflow. Each step shows a brief explanation of its purpose and the command to run it. Press **Enter** with no command to advance to the next step of the wizard. Use `wiz quit` to stop the wizard.

Wizards provide a tutorial and must be run in [Console](#) mode. Use the workflow command to run scripted workflow.

Command	Description	Usage
Curr	Shows the current step in the active workflow.	wizard curr
List	Lists the wizard workflow.	wizard list
Next	Go to the next step in the wizard	wizard next
Prev	Go to the previous step in the wizard	wizard prev
Quit	Stop and exit the wizard	wizard quit
Start	Start a wizard	wizard start [<workflowName>] <workflowName> - (required) Name of workflow to run Name can be abbreviated by truncation

Examples

To list the wizard workflow:

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

To start a wizard:

```
demo@ubuntu:~/shipped: shipped console
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>
```

To go to the next wizard step:

This command is not necessary to advance through a wizard, as simply pressing **Enter** with no input advances to the next step. The Shipped console displays a message when this option is available.

```
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>
```

To review the wizard step:

```
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.

To review previous wizard workflow steps:

```
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>
```

To stop the wizard before it completes:

The wizard workflow ends automatically when you complete the last step. However, you can terminate any time with the `wiz quit` command.

```
List available buildpacks with the "buildpack get" command.
```

```
bp get
```

```
Shipped> wiz q
```

Workflow

Workflow [wf] runs an orchestrated and repeatable set of commands.

A workflow is a number of CLI commands run sequentially. For example, you can run a workflow that creates a project with a service, then deploys it to the cloud.

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. Use `List` to see the current workflow commands.

Command	Description	Usage	Returns
Help	Shows help information for a workflow.	<code>wf help <workflowName> <workflowName> - (required) the name of the workflow</code>	Help text for the requested workflow
List	Lists all workflows or the contents of a single workflow.	<code>wf list <workflowName> <workflowName> - (optional) the name of the workflow</code>	All workflows or contents of a single workflow
Resume	In console mode, resume execution of a workflow from the point of an error.	<code>resume [--arg1 value1 [--arg2 value2...]] <-- help> <--verbose> <--arg1 value> - Zero or more arguments that override values saved when the original failed workflow run. <--help> - Show help for the workflow. <--verbose> - List each workflow command before execution</code>	Resumes execution of a failed workflow from the point of its failure
Run	Run a Shipped workflow.	<code>run <workflowName> [--arg1 value1 [-- arg2 value2...]] <- -help> <--verbose></code>	Output from the commands in the workflow

Examples

To show help for a workflow:

```
demo@ubuntu:~/shipped: wf help create-and-deploy
```

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

To show a list of workflows:


```
demo@ubuntu:~/shipped: wf list
```

To show contents of a workflow:

```
demo@ubuntu:~/shipped: wf list create-and-deploy
```

To run a workflow:

```
demo@ubuntu:~/shipped: wf run create-and-deploy --project hello-world --service cmx-service --framework  
cmx
```

Glossary

A

API

The Shipped API runs the same commands as using the interface. The API can be scripted.

B

build

A Git commit to a specific service.

buildpack

The starting point for a service, based on popular web frameworks.

C

Cisco Intercloud Services

The Cisco Intercloud Infrastructure as a Service (IaaS) that contains OpenStack plus many additional services.

CLI

The Shipped CLI lets you run Shipped commands through the terminal window.

config

Contains instructions to run a service.

D

deploy targets

Either cloud or on-premise clusters where you provision your Shipped services.

Docker

An OpenStack application that automates the deployment of a target. Docker makes the application run regardless of hardware, language, or hosting provider.

Drone

Triggers a build when it senses a commit being made on any branch.

E

environment

Manages namespace deployment, such as Development, Staging, or Production.

P

project

The container for the services used in your application.

R

release

A service deployment snapshot.

S

service

GitHub repositories that are part of a project.

U

users

End-users that have access to your project or service.

V

Vagrant

An OpenStack application that manages VMs and cloud instances. Commonly used to set up development or staging environments.