



First Ever CI/CD Debugger

Codefresh

DAN GARFIELD - KOSTIS KAPELONIS

Dan Garfield

Chief Technology Evangelist

 **codefresh**



Agenda

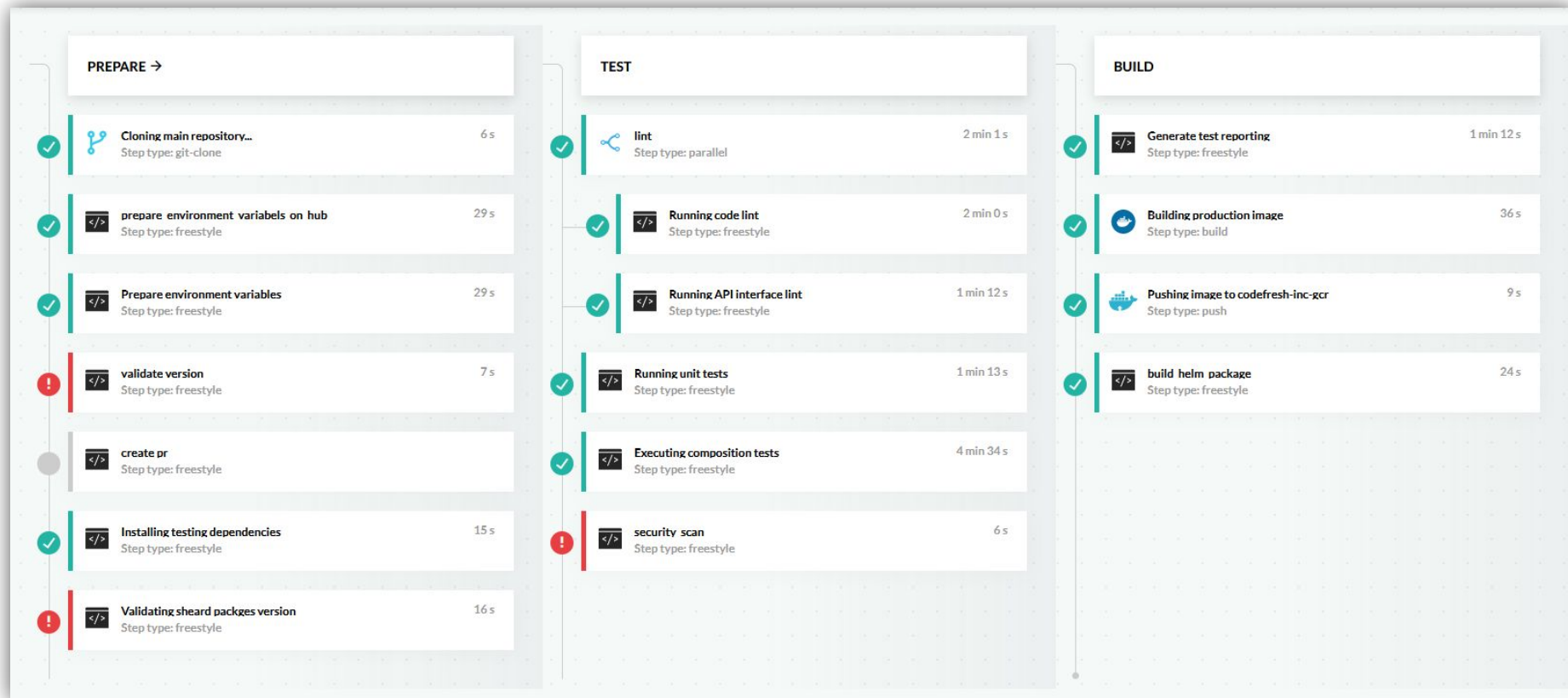
1. How debugging works in pipelines
2. Moving in/out of containers
3. Dynamic tool installation
4. Debugging and security
5. Debugging demos

About Codefresh

- Docker based CI/CD solution
- Each build step is a Docker image
- Native support for Docker, Helm, Kubernetes deployments
- Includes built-in Docker registry and Helm repository
- 20,000+ users

The screenshot displays the Codefresh web interface for a pipeline. On the left is a teal sidebar with navigation links: BUILD, TEST, DEPLOY, Pipelines, Builds, Monitoring, Kubernetes, Helm Releases, Docker Swarm, Images, Repositories, Helm Charts, Configuration, Account Settings, and User Settings. The main panel shows a pipeline named 'Pipeline Name' with a 'COMPLETED' status. It features a timeline with phases: 'INITIALIZATION', 'BUILD', and 'UNIT'. Each phase contains steps, all of which are 'GIT CLONE Cloning main repository'. A 'Run from here' button is visible over one of the build steps. The interface also includes a top navigation bar with 'Documentation', 'Support', and 'TRIGGER PIPELINE' buttons, and a bottom status bar with the text 'Select pipeline step to view details.'

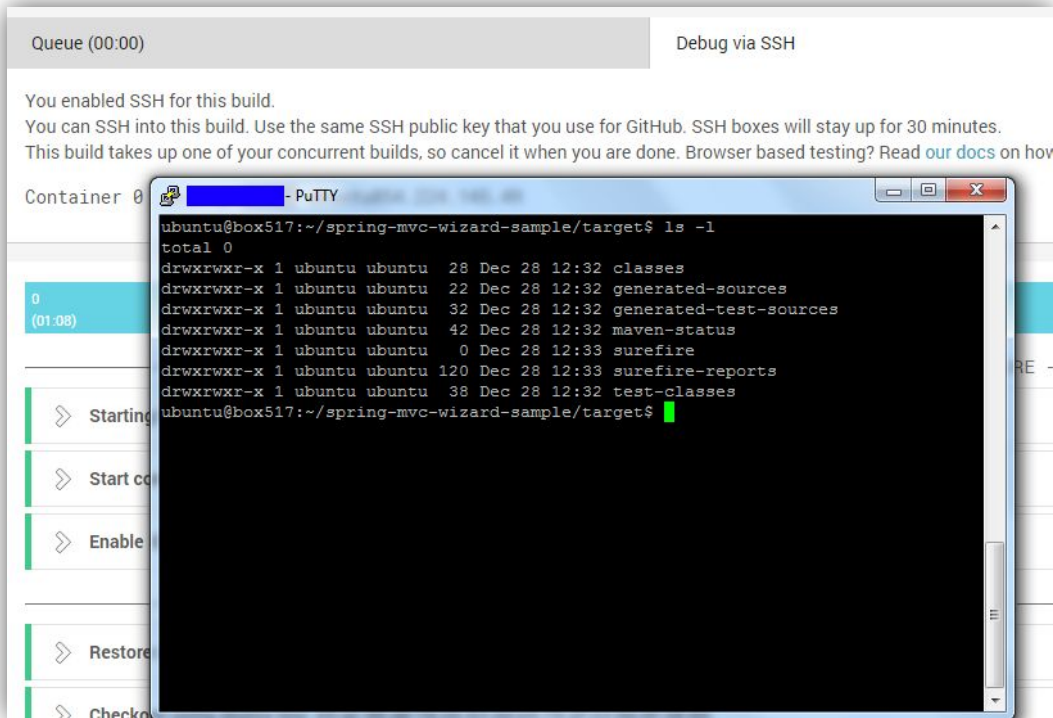
Pipelines break. It happens!



Other CI solutions

SSH is not very helpful

- Only works for legacy VM based pipelines
- Can only verify tools installed
- Lack of running context
- There is no state (e.g. environment variables)



CREATE

**DevOps = bringing developers
and operators together**

Debugging Code

The familiar experience

Place breaking points on code

```
91 // UnmarshalYAML is a custom unmarshaler that wraps strings in arrays
92 func (a *StringArray) UnmarshalYAML(unmarshal func(interface{}) error) error {
93     var strings []string
94     if err := unmarshal(&strings); err != nil {
95         var str string
96         if err := unmarshal(&str); err != nil {
97             return err
98         }
99         *a = []string{str}
100     } else {
101         *a = strings
102     }
103     return nil
104 }
105
106 // FlagArray is a wrapper for an array of strings
107 type FlagArray []string
108
109 // UnmarshalYAML is a custom unmarshaler that wraps strings in arrays
110 func (a *FlagArray) UnmarshalYAML(unmarshal func(interface{}) error) error {
111     var flags []string
112     if err := unmarshal(&flags); err != nil {
113         var flagstr string
114         if err := unmarshal(&flagstr); err != nil {
115             return err
```

View calls/
variables,
resume/step
forward

VARIABLES

- Local
 - Return value: undefined
 - this: global
 - next: function next(err) { ... }
 - productId: 1
 - req: IncomingMessage { _readableSt...
 - res: ServerResponse { domain: null...
- Closure
- Global

WATCH

- productId: 1

CALL STACK PAUSED ON STEP

(anonymous function)	.../routes/ind...
handle	layer.js 95:5
next	route.js 137:13
dispatch	route.js 112:3
handle	layer.js 95:5
(anonymous function)	.../router/ind...
process_params	.../router/index.js
next	.../router/index.js 275:10
handle	.../router/index.js 174:3
router	.../router/index.js 47:12
handle	layer.js 95:5
trim_prefix	.../router/index.js

BREAKPOINTS

- ☐ All Exceptions
- ☐ Uncaught Exceptions
- ☒ index.js routes 10
- ☒ index.js routes 12

index.js

```
1 var express = require('express');
2 var router = express.Router();
3
4 var fs = require('fs');
5
6 var Cart = require('../models/cart'); Cart = function Cart(cart) { ... }
7 var products = JSON.parse(fs.readFileSync('./data/products.json', 'utf8
8
9 router.get('/', function (req, res, next) { req = IncomingMessage { rea
10 var productId = products[0].id; products = Array(4) [Obj
11
12 res.render('index', res = ServerResponse { domain: null, _events: Obj
13 {
14   title: 'NodeJS Shopping Cart',
15   products: products products = Array(4) [Object, Object, Object, ...]
16 }
17 });
18 });
19
20 router.get('/add/:id', function(req, res, next) {
21
22   var productId = req.params.id;
23   var cart = new Cart(req.session.cart ? req.session.cart : {});
24   var product = products.filter(function(item) {
25     return item.id == productId;
26   });
27   cart.add(product[0], productId);
28   req.session.cart = cart;
29   res.redirect('/');
30   inline();
31 });
32
33 router.get('/cart', function(req, res, next) {
```

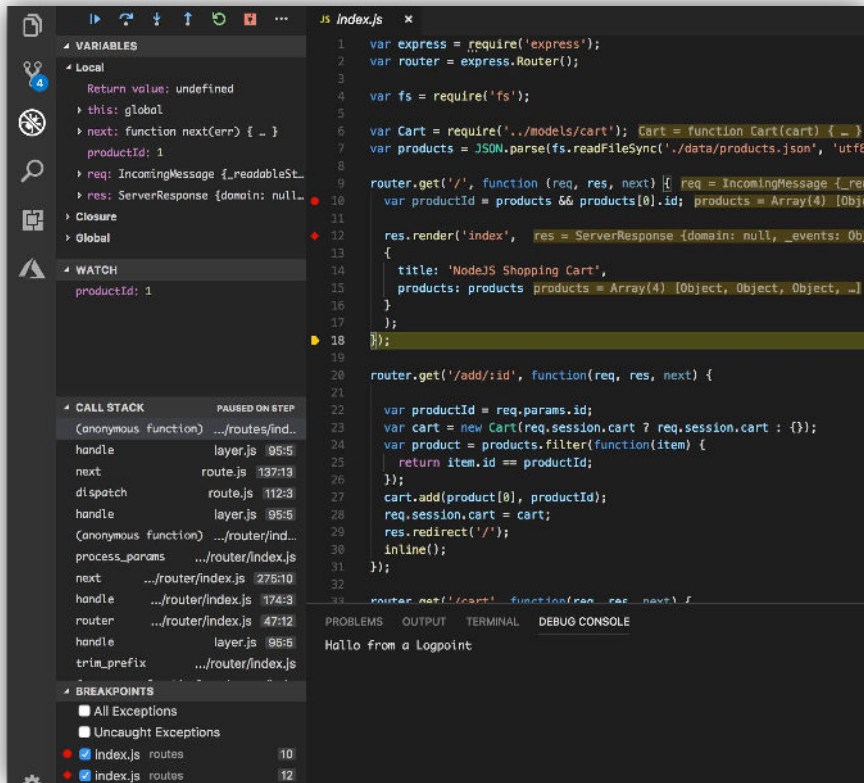
DEBUG CONSOLE

Hallo from a Logpoint

Live debugging

Quick isolation of errors

- View exact running state
- Pause/resume any line
- No printf's and printouts needed anymore



CREATE

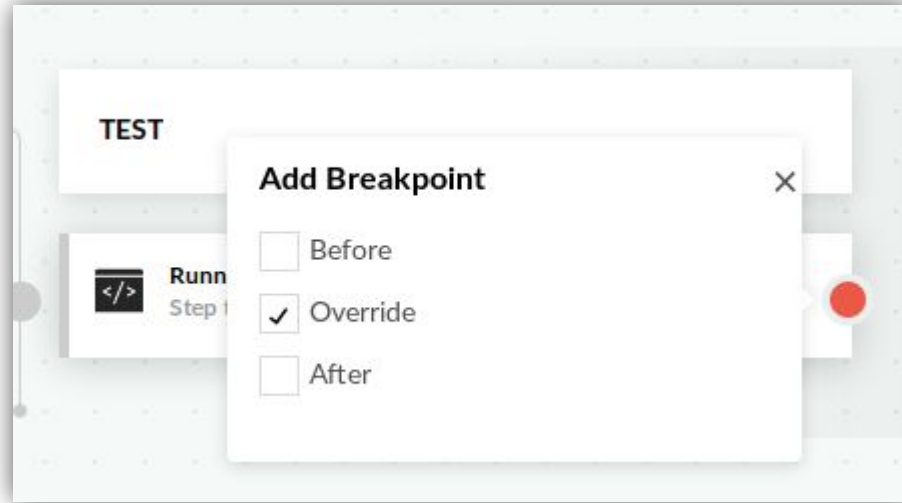
Developers

**We love IDEs
and live
debugging**

Operators

**We want live
debugging as
well!**

CI/CD Live Debugging



CI/CD live debugging

The screenshot displays a CI/CD pipeline interface with three main stages: **PREPARE**, **BUILD**, and **TEST**.

- PREPARE** (15 s): Includes the step "Cloning main repository..." (Step type: git-clone) with a green checkmark icon.
- BUILD** (0 s): Includes the step "Building Docker Image" (Step type: build) which is currently selected and highlighted with a blue border and a red target icon.
- TEST**: Includes the step "Running Unit tests" (Step type: freestyle) with a red circle icon.

Below the pipeline overview, the **Building Docker Image** step is expanded, showing a **DEBUG CONSOLE** with the following output:

```
-----
Debug Status: "before"
-----
bash-4.4# ls
cf_export          python-flask-sample-app  setup_debugger.sh
env_vars_to_export sensitive
bash-4.4# pwd
/codefresh/volume
bash-4.4# ls -l python-flask-sample-app/
.dockerrignore Dockerfile    codefresh.yml  setup.py
.git/          MANIFEST.in  minitwit/     tests/
.gitignore     README.MD    setup.cfg
bash-4.4# ls -l python-flask-sample-app/
```

At the bottom of the interface, there is a navigation bar with tabs for **Output**, **YAML**, **State**, and **Debugger**. On the right side of the expanded step, there are performance metrics for CPU, MEMORY, and LOG, all showing "N/A". Below these are buttons for **BEFORE**, **OVERWRITE**, **AFTER**, and **CONTINUE**. A chat bubble icon is visible in the bottom right corner.

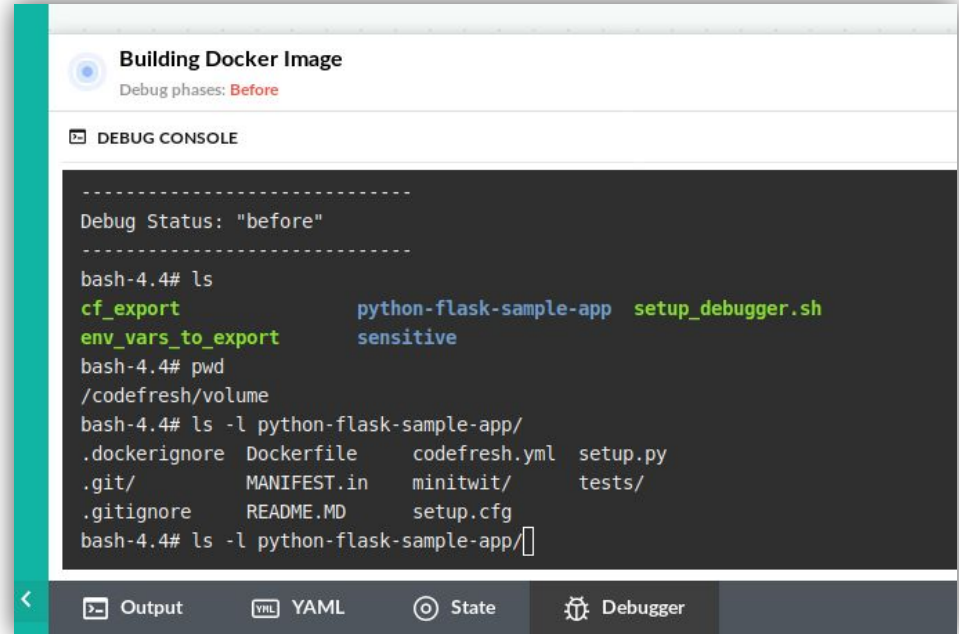
Demo: Simple Debugging

Demo: Container Tooling

Move in/out of containers

Install any tool

- Use Curl, nano, go, node
- Example: `apk add curl`
- All existing tools in your image



The screenshot shows a web-based interface for building a Docker image. At the top, it says "Building Docker Image" with a sub-label "Debug phases: Before". Below this is a "DEBUG CONSOLE" section containing a terminal window. The terminal shows the following commands and output:

```
-----  
Debug Status: "before"  
-----  
bash-4.4# ls  
cf_export          python-flask-sample-app  setup_debugger.sh  
env_vars_to_export sensitive  
bash-4.4# pwd  
/codefresh/volume  
bash-4.4# ls -l python-flask-sample-app/  
.dockerignore  Dockerfile      codefresh.yml  setup.py  
.git/           MANIFEST.in    minitwit/      tests/  
.gitignore     README.MD      setup.cfg  
bash-4.4# ls -l python-flask-sample-app/[]
```

At the bottom of the interface is a navigation bar with four tabs: "Output", "YAML", "State", and "Debugger". The "Output" tab is currently selected.

Demo: Dynamic Tools

Install tools on demand

Demo: Services

Access running services

What about Security?

Codefresh access controls

Control

- Pipelines
- Clusters
- From other repo

Permissions BACK TO APPLICATION

TEAM CAN Delete, Read, Update CLUSTERS THAT HAVE staging TAGS

sales TEAM CAN Read CLUSTERS THAT HAVE platform TAGS

Select Team ... TEAM CAN Any CLUSTERS THAT HAVE Any TAGS

[ADD RULE](#)

PIPELINES

sales TEAM CAN Approve, Read, Run, Update PIPELINES THAT HAVE all tags TAGS

Select Team ... TEAM CAN

[ADD RULE](#)

☒ Approve
☒ Read
☒ Run
☒ Update
☐ Create
☐ Debug
☐ Delete

[APPLY](#)

Pro debugging plan

- Breakpoints on any step
- Debug window
- Pause/Resume
- Standard commands (cd, ls, printenv etc)

Enterprise debugging plan

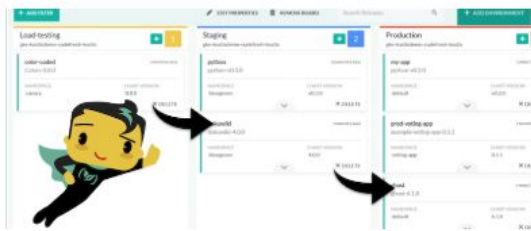
- Includes Pro plan
- Add extra tools dynamically (e.g. curl, wget, redis-cli)
- Shell auto-completion
- Restrict debug access

Summary

- World's first CI/CD debugger
- Place breakpoints on pipeline steps
- Stop/Resume pipeline
- Inspect files/variables/services
- Install tools on demand
- Lock-down debugging controls



About Kostis Kapelonis



Helm Tutorial | February 7, 2019

Using a Kanban board to manage and promote Helm Releases



Kostis Kapelonis

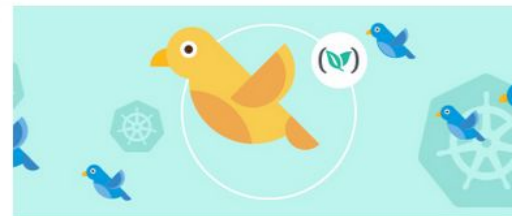


Continuous Integration | September 11, 2018

Programmatic Creation of Codefresh Pipelines – Part1



Kostis Kapelonis

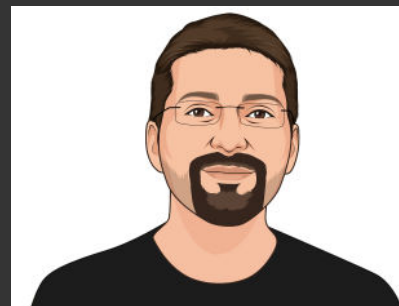


Continuous Deployment/Delivery | August 30, 2018

Fully automated canary deployments in Kubernetes



Kostis Kapelonis



Questions?

**Build Fast,
Deploy Faster**

Signup for a FREE account with
UNLIMITED builds

& schedule a 1:1 with
our experts at

<https://codefresh.io>

