This tutorial will show you how to create a production quality Kubernetes cluster!

We'll also show you how to setup a sample application that will build and deploy on every push to the Github master branch.

Index

- Creating the Kubernetes cluster
    - Getting the Kops tool
    - Creating a Key Pair
    - Creating the Kube cluster
        - Create the cluster configs
        - Get cluster
    - Edit the Cluster Configs
    - Get the current instance groups
        - Build the Cluster
- kubectl
    - Getting kubectl tool
    - Connection to the Kube API with the Kubectl tool
- General Kubernetes Usage
    - Listing Pods
- You have a Kubernetes cluster, now what?
    - Launching Ingress
- Build and Deploy With Wercker
    - Triggering your first build

- Configuring continuous deployments
- Conclusion

Creating the Kubernetes cluster

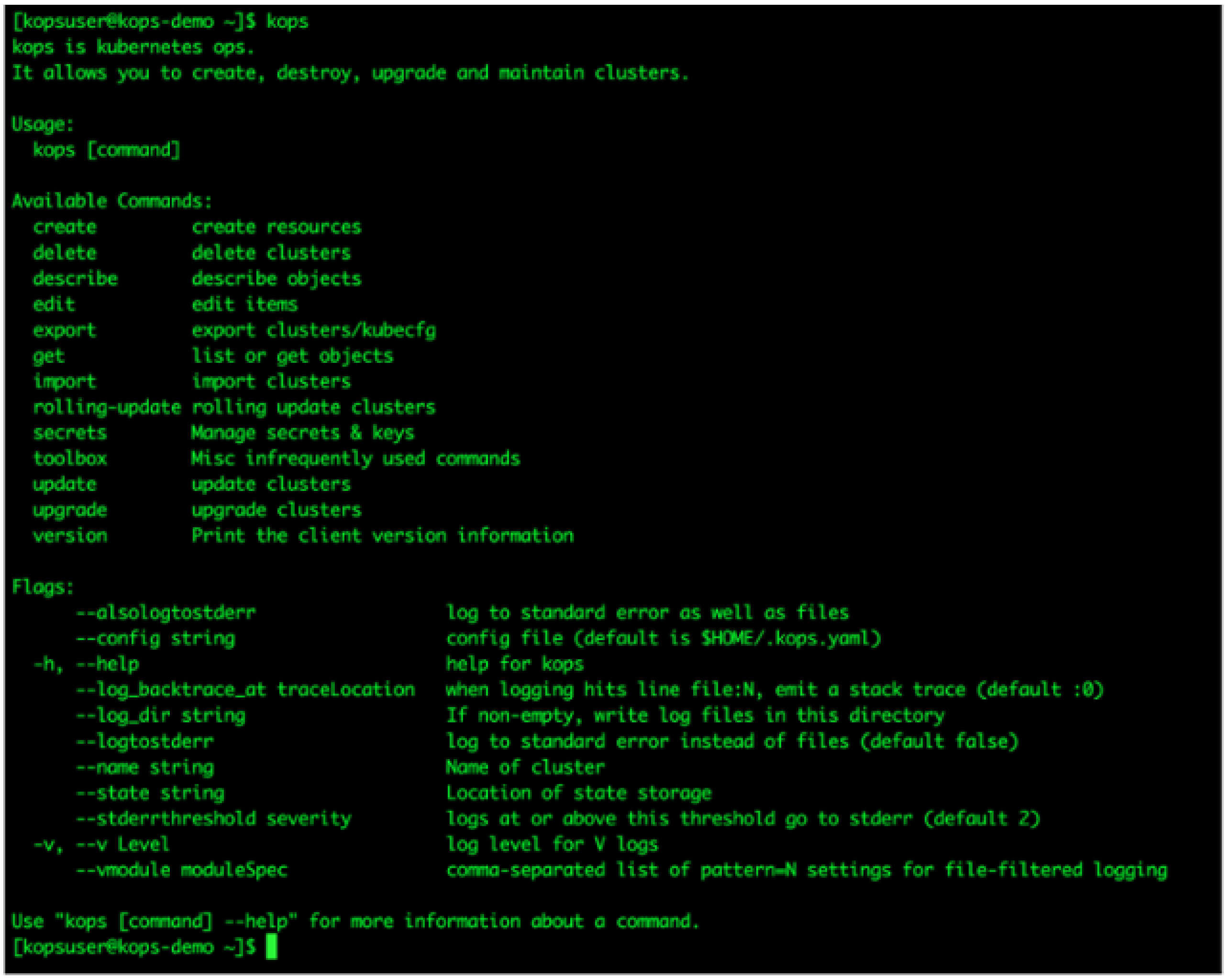
Getting the Kops tool

The list of the latest downloads for kops can be found here:

https://github.com/kubernetes/kops/releases/

```
$ chmod +x kops-linux-amd64
$ mv kops-linux-amd64 /usr/local/bin/
```

Confirm that it is properly installed by typing kops. You should get a similar output:



```
[kopsuser@kops-demo ~]$ kops
kops is kubernetes ops.
It allows you to create, destroy, upgrade and maintain clusters.

Usage:
  kops [command]

Available Commands:
  create         create resources
  delete         delete clusters
  describe       describe objects
  edit           edit items
  export         export clusters/kubecfg
  get            list or get objects
  import         import clusters
  rolling-update rolling update clusters
  secrets        Manage secrets & keys
  toolbox        Misc infrequently used commands
  update         update clusters
  upgrade        upgrade clusters
  version        Print the client version information

Flags:
      --alsologtostderr                  log to standard error as well as files
      --config string                    config file (default is $HOME/.kops.yaml)
  -h, --help                             help for kops
      --log_backtrace_at traceLocation   when logging hits line file:N, emit a stack trace (default :0)
      --log_dir string                   If non-empty, write log files in this directory
      --logtostderr                      log to standard error instead of files (default false)
      --name string                      Name of cluster
      --state string                     Location of state storage
      --stderrthreshold severity         logs at or above this threshold go to stderr (default 2)
  -v, --v Level                          log level for V logs
      --vmodule moduleSpec               comma-separated list of pattern=N settings for file-filtered logging

Use "kops [command] --help" for more information about a command.
[kopsuser@kops-demo ~]$
```

Creating a Key Pair

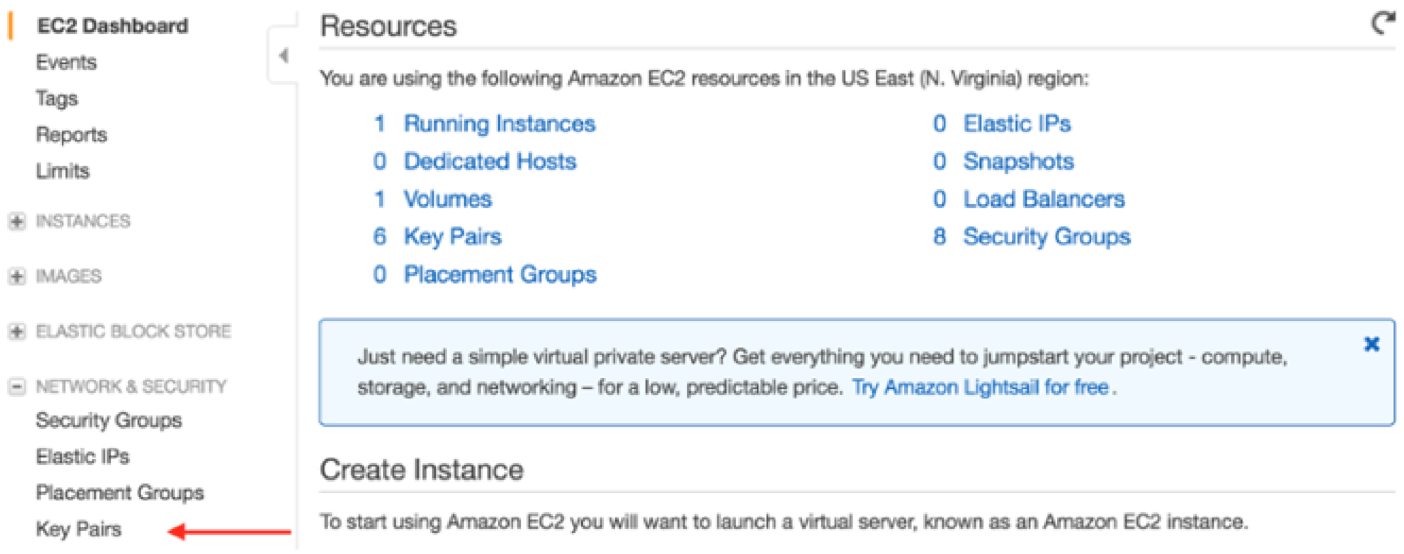
Please note: you will need to create your own ssh public key and import it to the Key Pairs section under EC2 in the Amazon Web Console.

Before we create our kube cluster we will need to create our SSH key pair needed for access:

1. Log into your AWS web console.

2. Select the EC2 option.

3. Under the Network & Security option select Key Pairs.



Resources

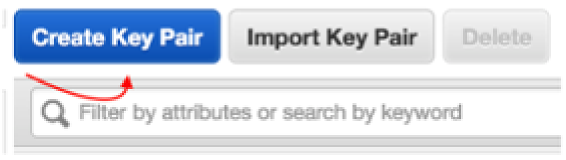
You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

1	Running Instances	0	Elastic IPs
0	Dedicated Hosts	0	Snapshots
1	Volumes	0	Load Balancers
6	Key Pairs	8	Security Groups
0	Placement Groups		

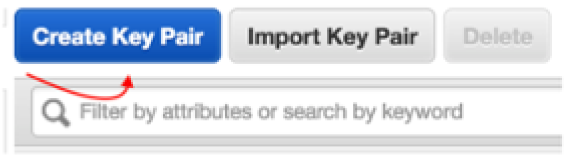
Just need a simple virtual private server? Get everything you need to jumpstart your project - compute, storage, and networking – for a low, predictable price. Try Amazon Lightsail for free.

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

4. Select Create Key Pair.



5. Give your Key Pair a name and click create.



The private key file is automatically downloaded by your browser. The filename is the name you specified as the name of your key pair, and the filename extension is .pem. **Save the private key file in a safe place!**

Important: This is the only chance for you to save the private key file. You'll need to provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.

6. If you use an SSH client on a Mac or Linux computer to connect to your Linux instance, use the following command to set the permissions of your private key file so that only you can read it:

```
$ chmod 400 my-key-pair.pem
```

Creating the Kube cluster

Please Note: Before we can create our kube cluster we will have to:

- Create a route53 domain for our cluster.
- Create an S3 bucket to store our clusters state with the following name: kop-store-**<your_name>**.

Instructions for accomplishing both can be found at steps 2 and 3 at the link below:

https://kubernetes.io/docs/getting-started-guides/kops/

Export the various env variables:

```
export AWS_ACCESS_KEY_ID=""
export AWS_SECRET_ACCESS_KEY="
export AWS_DEFAULT_REGION=us-east-1
export KOPS_STATE_STORE=s3://kop-store-<your_name>
export NAME=demo.k8s.<your_valid_domain_name>
Example of a valid domain name is: demo.k8s.wercker.com
```

Create the cluster configs:

```
 kops create cluster
--name=${NAME}
--cloud=aws
--kubernetes-version=1.5.1
--zones=us-east-1b,us-east-1c,us-east-1d
--ssh-public-key=my-key-pair.pem
--network-cidr=172.30.0.0/16
--admin-access=172.30.0.0/16
--node-count=1
--master-zones=us-east-1b,us-east-1c,us-east-1d
--master-size=t2.medium
```

Get cluster

The below command will show you your newly created cluster:

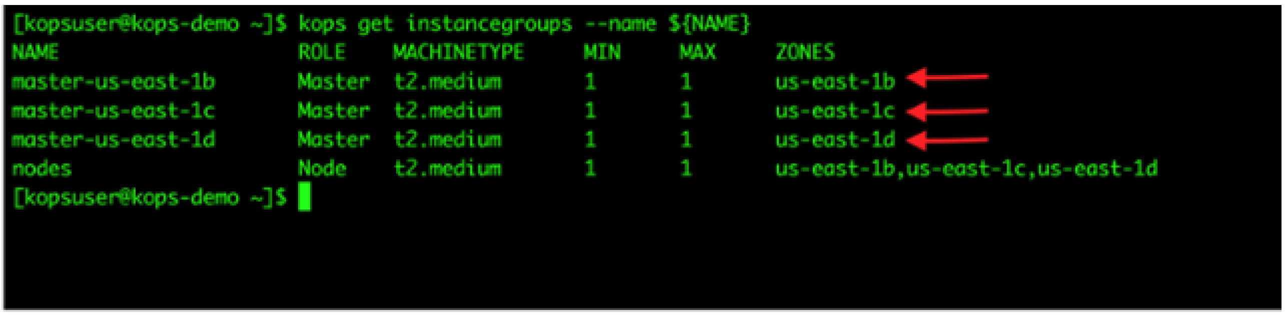
```
 $ kops get cluster
NAME   CLOUD    ZONES
Demo.k8s.wercker.com  aws        us-east-1b
```

Edit the Cluster Configs:

```
kops edit cluster ${NAME}
```

Get the current instance groups:

```
$ kops get instancegroups --name ${NAME}
```



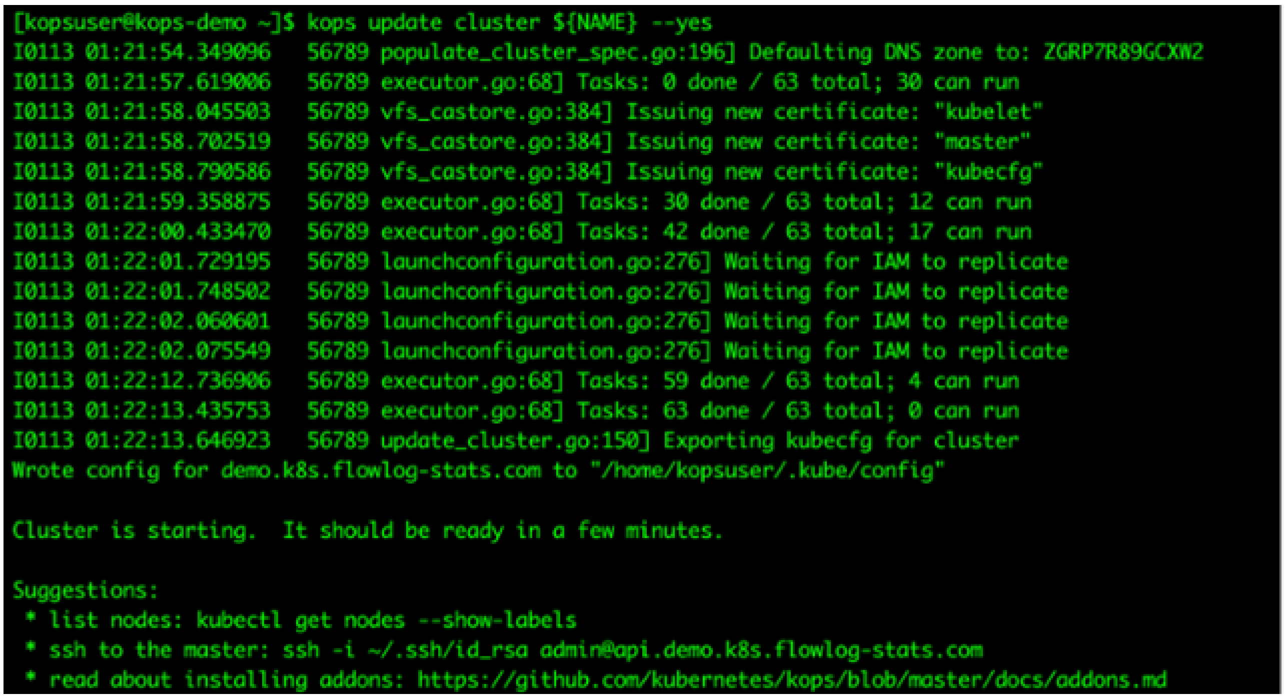
```
[kopsuser@kops-demo ~]$ kops get instancegroups --name ${NAME}
NAME                    ROLE    MACHINETYPE     MIN     MAX     ZONES
master-us-east-1b       Master  t2.medium       1       1       us-east-1b
master-us-east-1c       Master  t2.medium       1       1       us-east-1c
master-us-east-1d       Master  t2.medium       1       1       us-east-1d
nodes                   Node    t2.medium       1       1       us-east-1b,us-east-1c,us-east-1d
[kopsuser@kops-demo ~]$
```

This output shows you that it will create 4 nodes all together. There will be 3 master nodes for redundancy and 1 minion node.

Build the Cluster

This will start launching: subnets, ec2 instances, and start creating your cluster:

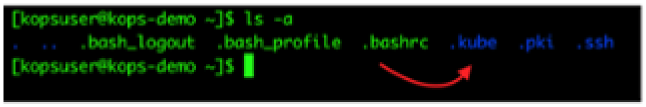
```
kops update cluster ${NAME} --yes
```



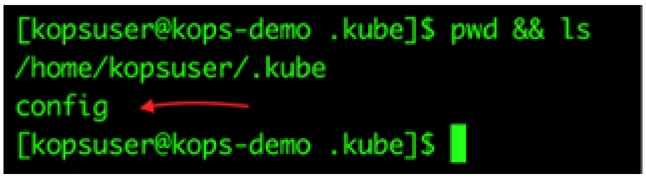
```
[kopsuser@kops-demo ~]$ kops update cluster ${NAME} --yes
I0113 01:21:54.349096   56789 populate_cluster_spec.go:196] Defaulting DNS zone to: ZGRP7R89GCXW2
I0113 01:21:57.619006   56789 executor.go:68] Tasks: 0 done / 63 total; 30 can run
I0113 01:21:58.045503   56789 vfs_castore.go:384] Issuing new certificate: "kubelet"
I0113 01:21:58.702519   56789 vfs_castore.go:384] Issuing new certificate: "master"
I0113 01:21:58.790586   56789 vfs_castore.go:384] Issuing new certificate: "kubecfg"
I0113 01:21:59.358875   56789 executor.go:68] Tasks: 30 done / 63 total; 12 can run
I0113 01:22:00.433470   56789 executor.go:68] Tasks: 42 done / 63 total; 17 can run
I0113 01:22:01.729195   56789 launchconfiguration.go:276] Waiting for IAM to replicate
I0113 01:22:01.748502   56789 launchconfiguration.go:276] Waiting for IAM to replicate
I0113 01:22:02.060601   56789 launchconfiguration.go:276] Waiting for IAM to replicate
I0113 01:22:02.075549   56789 launchconfiguration.go:276] Waiting for IAM to replicate
I0113 01:22:12.736906   56789 executor.go:68] Tasks: 59 done / 63 total; 4 can run
I0113 01:22:13.435753   56789 executor.go:68] Tasks: 63 done / 63 total; 0 can run
I0113 01:22:13.646923   56789 update_cluster.go:150] Exporting kubecfg for cluster
Wrote config for demo.k8s.flowlog-stats.com to "/home/kopsuser/.kube/config"

Cluster is starting.  It should be ready in a few minutes.

Suggestions:
 * list nodes: kubectl get nodes --show-labels
 * ssh to the master: ssh -i ~/.ssh/id_rsa admin@api.demo.k8s.flowlog-stats.com
 * read about installing addons: https://github.com/kubernetes/kops/blob/master/docs/addons.md
```

Once you build the cluster the kube config file with the authorization credentials for access will be in your home directory:

```
~/.kube/config
```



```
[kopsuser@kops-demo ~]$ ls -a
.  ..  .bash_logout  .bash_profile  .bashrc  .kube  .pki  .ssh
[kopsuser@kops-demo ~]$
```



```
[kopsuser@kops-demo .kube]$ pwd && ls
/home/kopsuser/.kube
config
[kopsuser@kops-demo .kube]$
```

kubectl

Getting kubectl tool

Download the kubectl command line tool:

```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://stora
```

Make the kubectl binary executable and move it to your PATH:

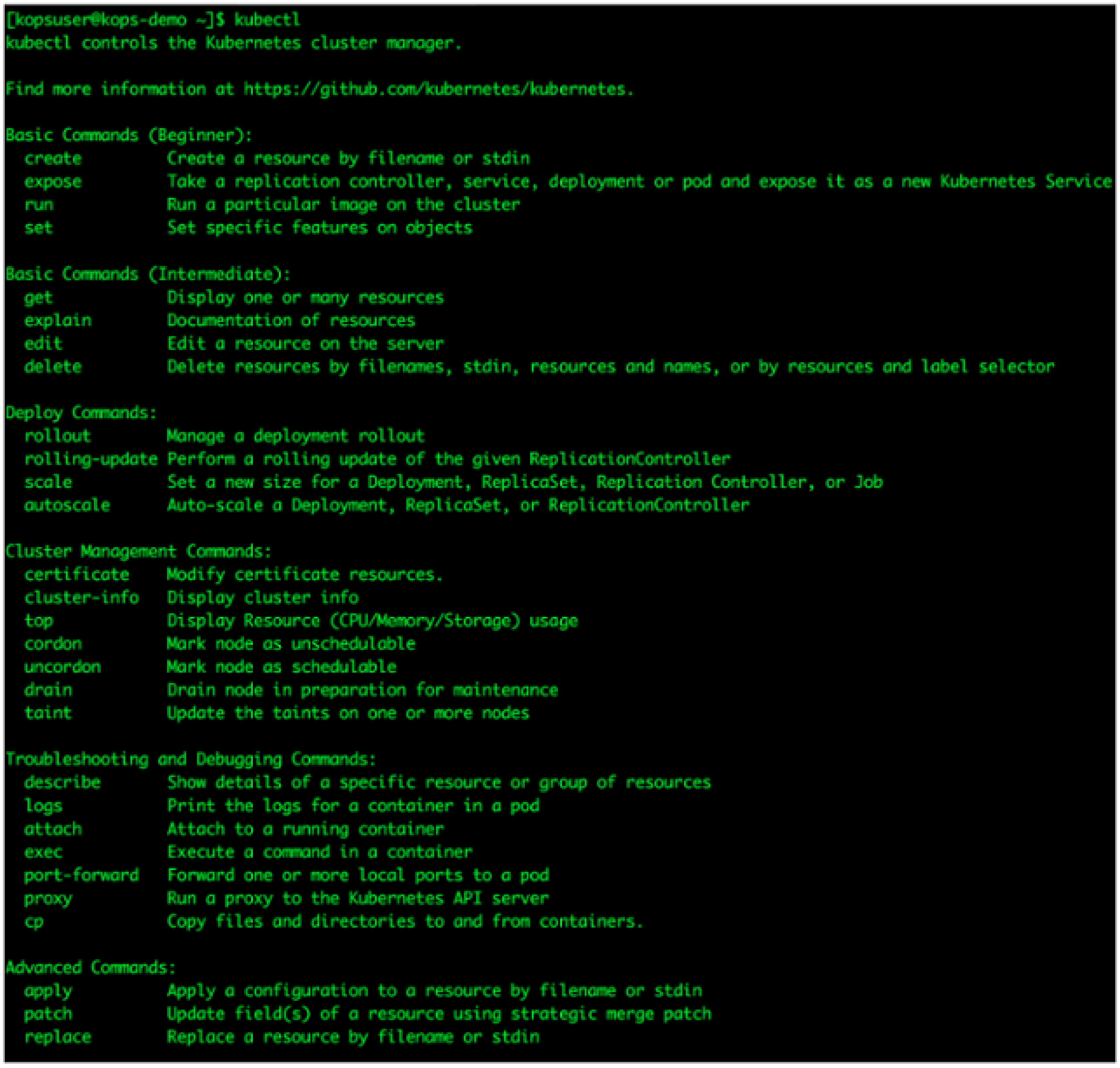
```
$ chmod +x kubectl
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

After downloading let's enable shell autocompletion:

```
$ echo "source <(kubectl completion bash)" >> ~/.bashrc
```

Once you've moved the kubectl binary to `/usr/local/bin/` confirm that it is properly installed by typing the below command. You should get a similar output from the screen capture below:

```
$ kubectl
```



```
[kopsuser@kops-demo ~]$ kubectl
kubectl controls the Kubernetes cluster manager.

Find more information at https://github.com/kubernetes/kubernetes.

Basic Commands (Beginner):
  create         Create a resource by filename or stdin
  expose         Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout        Manage a deployment rollout
  rolling-update Perform a rolling update of the given ReplicationController
  scale          Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
  autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes

Troubleshooting and Debugging Commands:
  describe       Show details of a specific resource or group of resources
  logs           Print the logs for a container in a pod
  attach         Attach to a running container
  exec           Execute a command in a container
  port-forward   Forward one or more local ports to a pod
  proxy          Run a proxy to the Kubernetes API server
  cp             Copy files and directories to and from containers.

Advanced Commands:
  apply          Apply a configuration to a resource by filename or stdin
  patch          Update field(s) of a resource using strategic merge patch
  replace        Replace a resource by filename or stdin
```

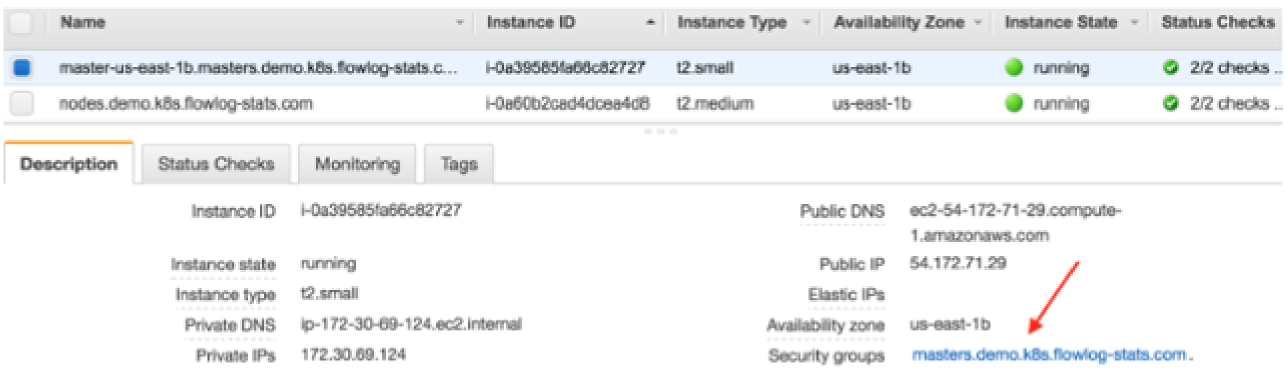
Connection to the Kube API with the Kubectl tool

The kubectl tool uses a kube config file (default location is in `~/.kube/config`) to connect and to communicate with our kube cluster. The file that Kops generates is pointing to the public IP address of the cluster.

We have set the security groups within AWS so that you cannot reach the public IP of the kube API. With that, you have to open up access to the kube API. **We want to severely limit access to these servers.**

At this point we will need to add our public IP address to the inbound rule for our master instance in the Security Group of our AWS console.

- Log into your AWS Console and select EC2.
- Select the master instance and click on its Security Group option.

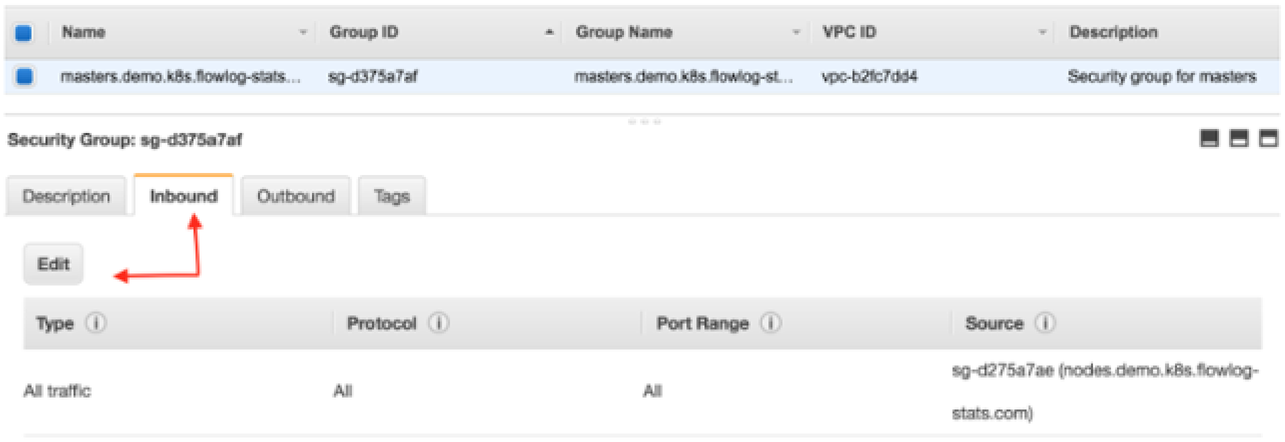


	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
■	master-us-east-1b.masters.demo.k8s.flowlog-stats.c...	i-0a39585fa66c82727	t2.small	us-east-1b	● running	✓ 2/2 checks ...
□	nodes.demo.k8s.flowlog-stats.com	i-0a60b2cad4dcea4d8	t2.medium	us-east-1b	● running	✓ 2/2 checks ...

Description | Status Checks | Monitoring | Tags

Instance ID	i-0a39585fa66c82727	Public DNS	ec2-54-172-71-29.compute-1.amazonaws.com
Instance state	running	Public IP	54.172.71.29
Instance type	t2.small	Elastic IPs	
Private DNS	ip-172-30-69-124.ec2.internal	Availability zone	us-east-1b
Private IPs	172.30.69.124	Security groups	masters.demo.k8s.flowlog-stats.com .

- Select the Inbound tab then Edit.



	Name	Group ID	Group Name	VPC ID	Description
■	masters.demo.k8s.flowlog-stats...	sg-d375a7af	masters.demo.k8s.flowlog-st...	vpc-b2fc7dd4	Security group for masters

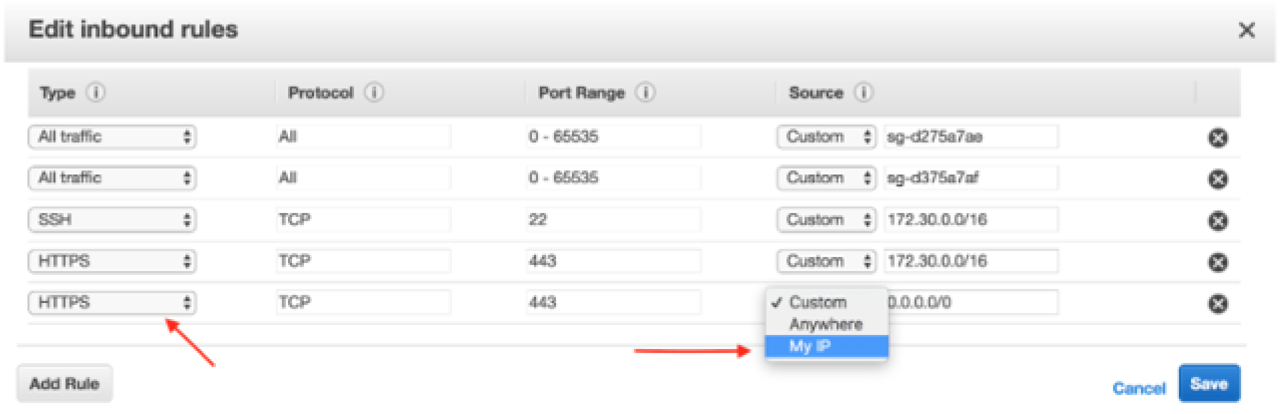
Security Group: sg-d375a7af

Description | Inbound | Outbound | Tags

Edit

Type	Protocol	Port Range	Source
All traffic	All	All	sg-d275a7ae (nodes.demo.k8s.flowlog-stats.com)

- Choose HTTPS for the type and select My IP then save.



Edit inbound rules

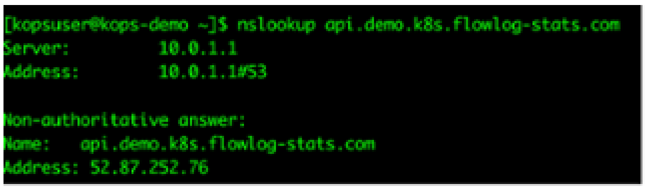
Type	Protocol	Port Range	Source	
All traffic	All	0 - 65535	Custom	sg-d275a7ae
All traffic	All	0 - 65535	Custom	sg-d375a7af
SSH	TCP	22	Custom	172.30.0.0/16
HTTPS	TCP	443	Custom	172.30.0.0/16
HTTPS	TCP	443	✓ Custom / Anywhere / My IP	0.0.0.0/0

- Before we leave the Security Group section let us repeat the steps above and whitelist the below IP addresses. We will need these IP's for one of our steps moving forward that allows the Wercker.com CI/CD service to reach our Kubernetes cluster.

After you've whitelisted your IP verify connectivity by checking with the nslookup command:

```
$ nslookup api.demo.k8s.wercker.com
```

Please Note: This can take up to 15 minutes for the DNS records to update with a response.



```
[kopsuser@kops-demo ~]$ nslookup api.demo.k8s.flowlog-stats.com
Server:         10.0.1.1
Address:        10.0.1.1#53

Non-authoritative answer:
Name:   api.demo.k8s.flowlog-stats.com
Address: 52.87.252.76
```

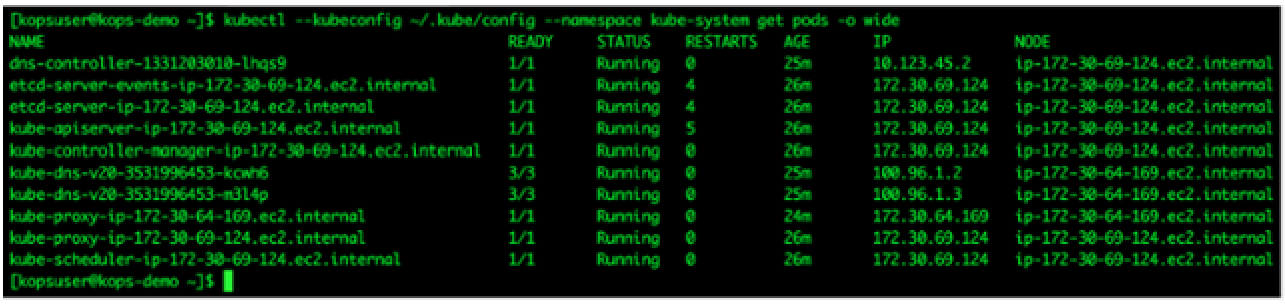
General Kubernetes Usage

Listing Pods

You can tell the command line which namespace you want to look at.

Another interesting namespace is the **kube-system** namespace. This is a namespace that is automatically created for you and it runs Kubernetes system pods in it.

```
kubectl --namespace kube-system get pods -o wide
```



```
[kopsuser@kops-demo ~]$ kubectl --kubeconfig ~/.kube/config --namespace kube-system get pods -o wide
NAME                                                  READY     STATUS    RESTARTS   AGE       IP              NODE
dns-controller-1331203010-lhqs9                       1/1       Running   0          25m       10.123.45.2     ip-172-30-69-124.ec2.internal
etcd-server-events-ip-172-30-69-124.ec2.internal      1/1       Running   4          26m       172.30.69.124   ip-172-30-69-124.ec2.internal
etcd-server-ip-172-30-69-124.ec2.internal             1/1       Running   4          26m       172.30.69.124   ip-172-30-69-124.ec2.internal
kube-apiserver-ip-172-30-69-124.ec2.internal          1/1       Running   5          26m       172.30.69.124   ip-172-30-69-124.ec2.internal
kube-controller-manager-ip-172-30-69-124.ec2.internal 1/1       Running   0          26m       172.30.69.124   ip-172-30-69-124.ec2.internal
kube-dns-v20-3531996453-kcwh6                         3/3       Running   0          25m       100.96.1.2      ip-172-30-64-169.ec2.internal
kube-dns-v20-3531996453-m3l4p                         3/3       Running   0          25m       100.96.1.3      ip-172-30-64-169.ec2.internal
kube-proxy-ip-172-30-64-169.ec2.internal              1/1       Running   0          24m       172.30.64.169   ip-172-30-64-169.ec2.internal
kube-proxy-ip-172-30-69-124.ec2.internal              1/1       Running   0          26m       172.30.69.124   ip-172-30-69-124.ec2.internal
kube-scheduler-ip-172-30-69-124.ec2.internal          1/1       Running   0          26m       172.30.69.124   ip-172-30-69-124.ec2.internal
[kopsuser@kops-demo ~]$
```

Also notice there is a -o wide at the end of the command. That is optional but it gives extra information on which node the pod is running on.

You have a Kubernetes cluster, now what?

We now need to launch some stuff into the cluster and we will use the below repo as an example:

https://github.com/wercker/kubernetes-ci-cd

First clone the folder to your system:

```
$ git clone https://github.com/wercker/kubernetes-ci-cd
$ cd kubernetes-ci-cd
```

Create the files needed with the kubectl create -f command for each of the yaml files below:

```
$ kubectl --namespace default create -f ./kubernetes/ingress/
```

This will create all of the resources in the `ingress` directory.

Launching Ingress

The Kubernetes Ingress is a Nginx pod that reads from the Kubernetes API to create it's own configuration so that it can dynamically route traffic inbound to the cluster.

The topology is as follows:

```
Internet<-->ELB<-->Ingress Nginx Pod<-->Application Pods
```

The ingress is created after the creation of the default-backend.yaml and rc.yaml files.

Verify the creation of the ingress (nginx) with:

```
 $ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
ingress-controller-2171626210-2bg8c        1/1       Running   0          4m
ingress-default-backend-2249275363-8f0tw   1/1       Running   0          4m
```

Build and Deploy With Wercker

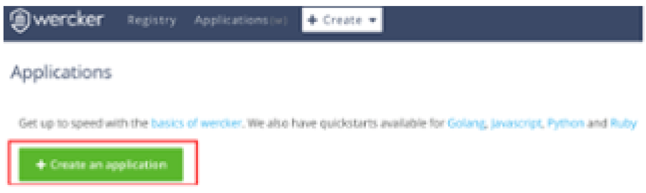
You can follow this step-by-step guide on developing, building and deploying a sample app with Wercker:

- http://devcenter.wercker.com/docs/quickstarts/building/golang

For our purposes we will walk through adding your app to Wercker:

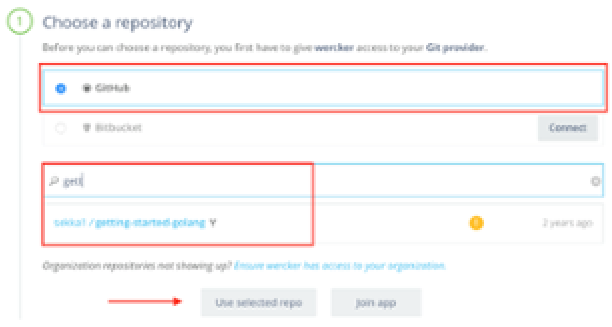
- Sign up for a wercker.com account (you can also sign in with your GitHub account)

- After signing in select the option to create a new Application.



Get up to speed with the basics of wercker. We also have quickstarts available for Golang, Javascript, Python and Ruby

- Select your Git provider along with the App we will deploy and lastly use selected repo.



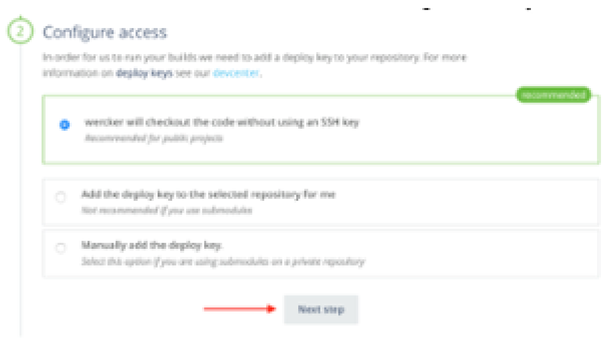
Before you can choose a repository, you first have to give wercker access to your Git provider.

get

sekka1 / getting-started-golang 2 years ago

Organization repositories not showing up? Ensure wercker has access to your organization.

- Choose Wercker will checkout the code without using an SSH key.



2 Configure access

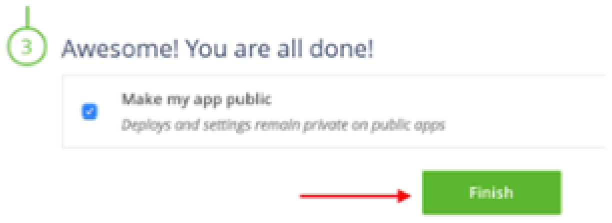
In order for us to run your builds we need to add a deploy key to your repository. For more information on deploy keys see our devcenter.

[✓ wercker will checkout the code without using an SSH key
Recommended for public projects]

Add the deploy key to the selected repository for me
Not recommended if you use submodules

Manually add the deploy key.
Select this option if you are using submodules on a private repository

- Check the box and select Finish!



3 Awesome! You are all done!

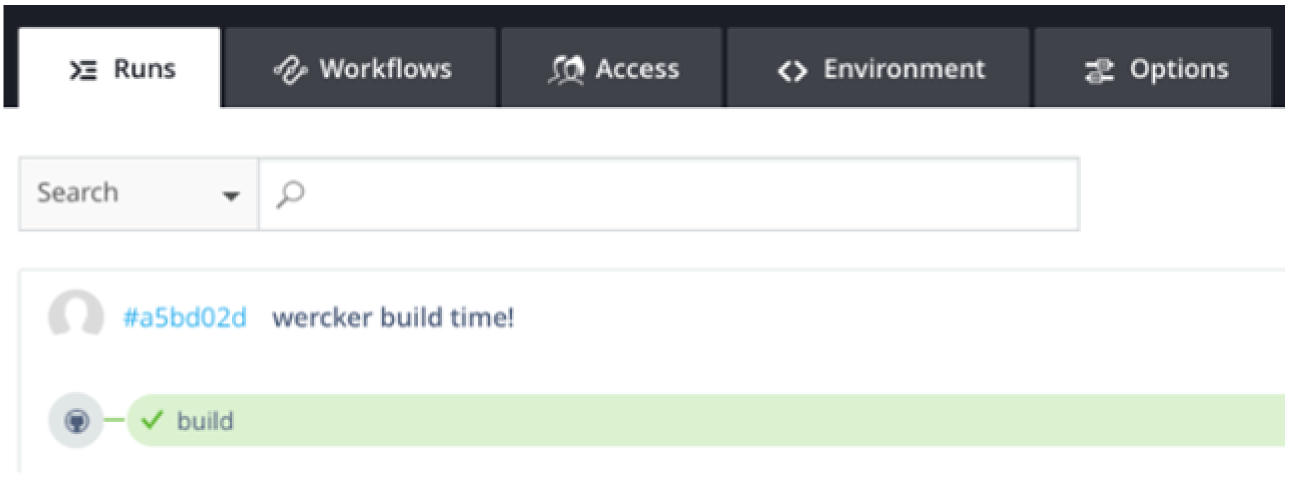
[✓ Make my app public]
Deploys and settings remain private on public apps

Triggering your first build

- Make a change to one of the files in the getting-started-golang folder from the command line then run the below commands:

```
$ git commit -am 'wercker build time!'
$ git push origin master
```

Finally, navigate to your app page where you will see a new build has been triggered!

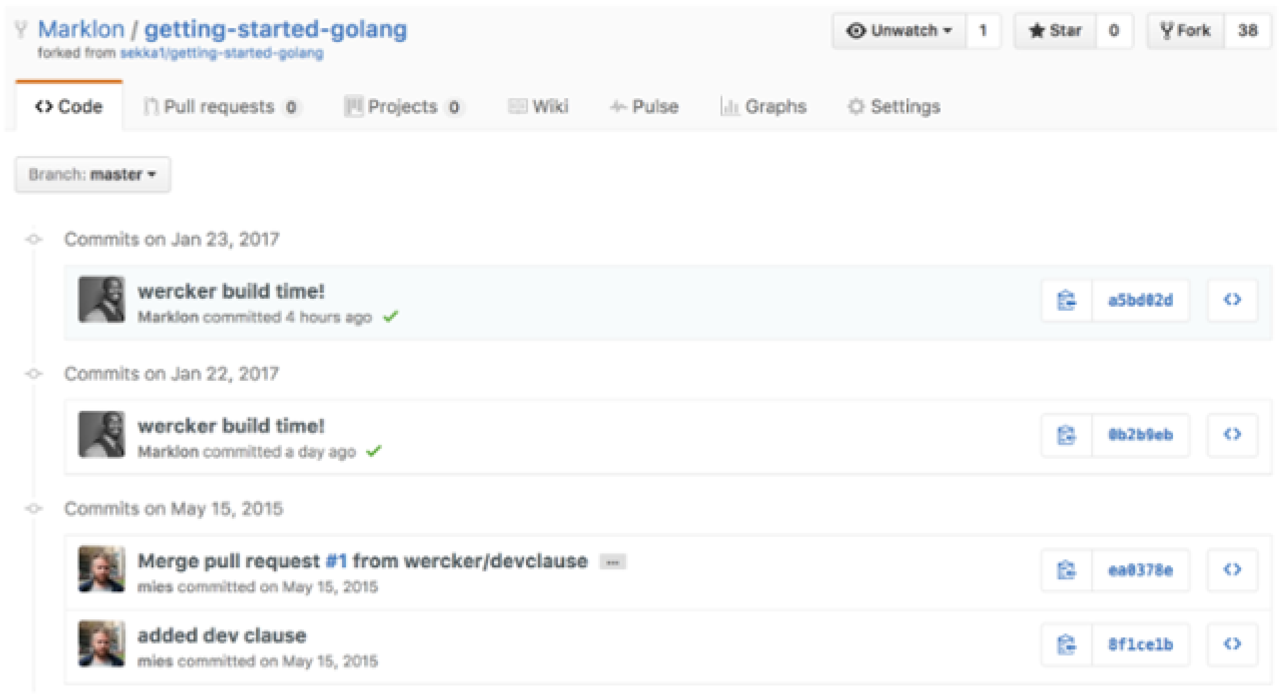


#a5bd02d wercker build time!

✓ build

You can also see the commits from wercker in the GitHub repo:

https://github.com/Marklon/getting-started-golang



Commits on Jan 23, 2017

wercker build time!
Marklon committed 4 hours ago ✓ a5bd02d

Commits on Jan 22, 2017

wercker build time!
Marklon committed a day ago ✓ 0b2b9eb

Commits on May 15, 2015

Merge pull request #1 from wercker/devclause
mies committed on May 15, 2015 ea0378e

added dev clause
mies committed on May 15, 2015 8f1ce1b

Configuring continuous deployments

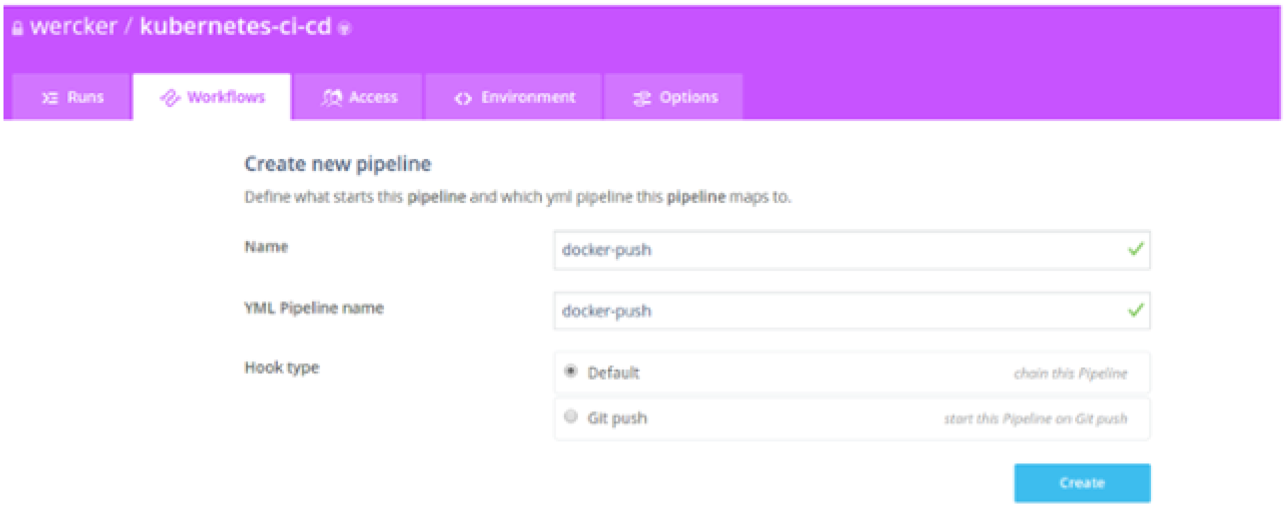
We will add 2 actions into Wercker.com that is in our wercker.yml file.

1) Push the built docker container to our quay.io repository. You can also push it to any other Docker repositories.

a) Our definition for the "push docker" action: https://github.com/wercker/kubernetes-ci-cd/blob/master/wercker.yml#L42.

Configure Wercker.com GUI for this new pipeline:

- Go to the “Workflow” tab and click the “Add new pipeline” button.
- Name: docker-push.
- YML pipeline name: docker-push.



Create new pipeline

Define what starts this **pipeline** and which yml pipeline this **pipeline** maps to.

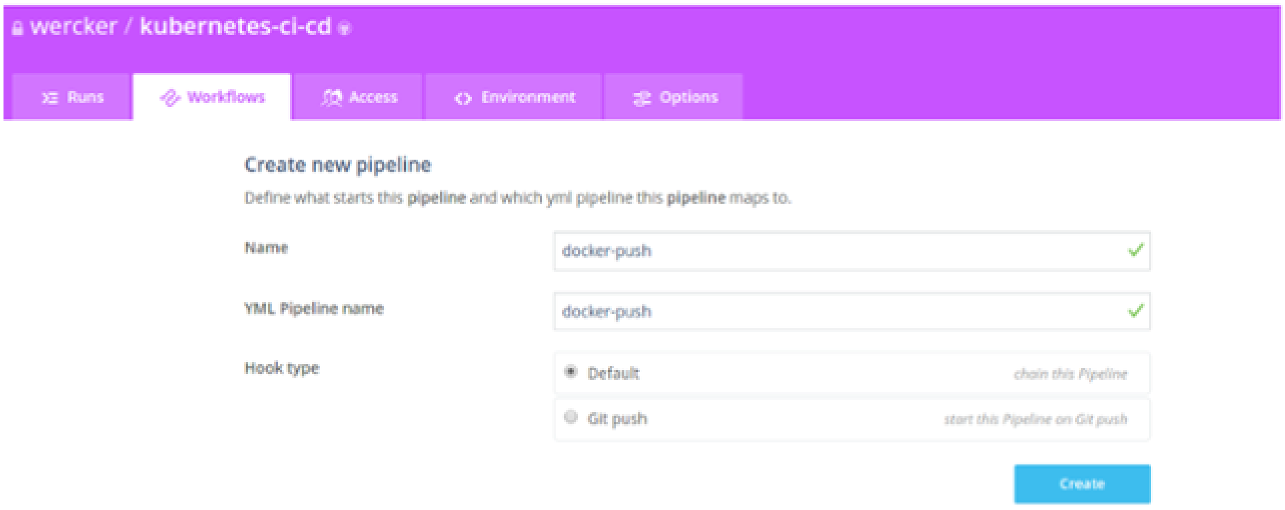
Name: docker-push

YML Pipeline name: docker-push

Hook type: [✓ Default] chain this Pipeline / Git push start this Pipeline on Git push

Configure Wercker.com GUI with our Quay.io credentials:

- Go to the “Environment” tab.
- Add a variable with key: QUAY_USERNAME.
    - This is a username with permission to your quay.io repository.
- Add a variable with key: QUAY_PASSWORD.
    - This is the password for the user.



wercker / kubernetes-ci-cd

Create new pipeline

Define what starts this **pipeline** and which yml pipeline this **pipeline** maps to.

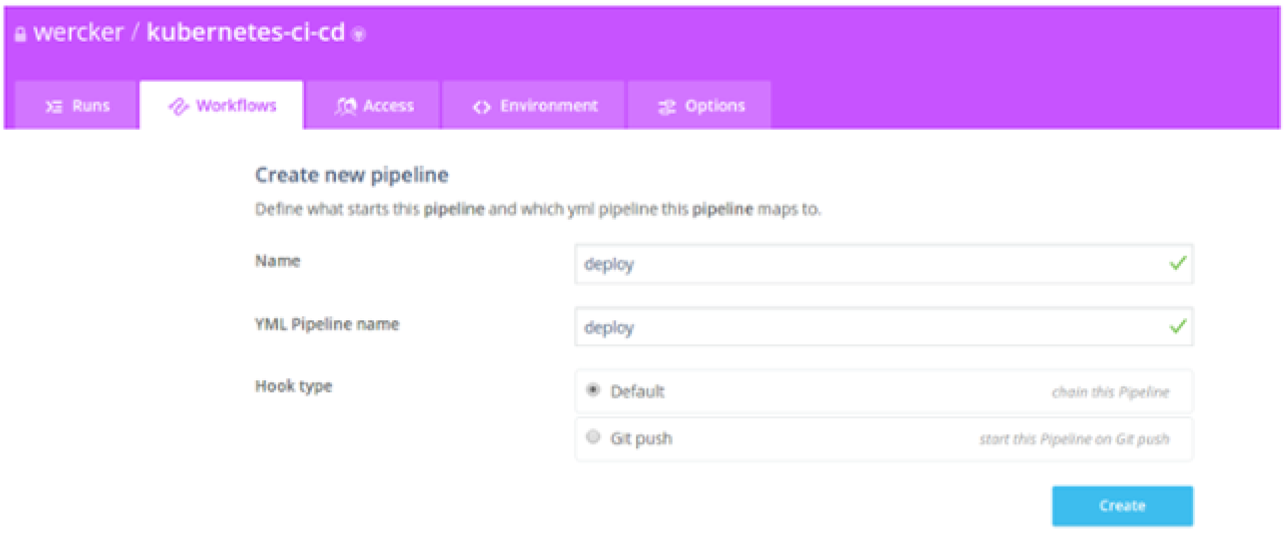
Name	docker-push
YML Pipeline name	docker-push
Hook type	[✓ Default] chain this Pipeline; Git push start this Pipeline on Git push

2) Deploy the built container onto our Kubernetes cluster when there is a push into the `master` branch.

a) Our definition for the "deploy" action: https://github.com/wercker/kubernetes-ci-cd/blob/master/wercker.yml#L72.

Configure Wercker.com GUI for this new pipeline:

- Go to the "Workflow" tab and click the "Add new pipeline" button.
- Name: deploy.
- YML pipeline name: deploy.



wercker / kubernetes-ci-cd

Create new pipeline

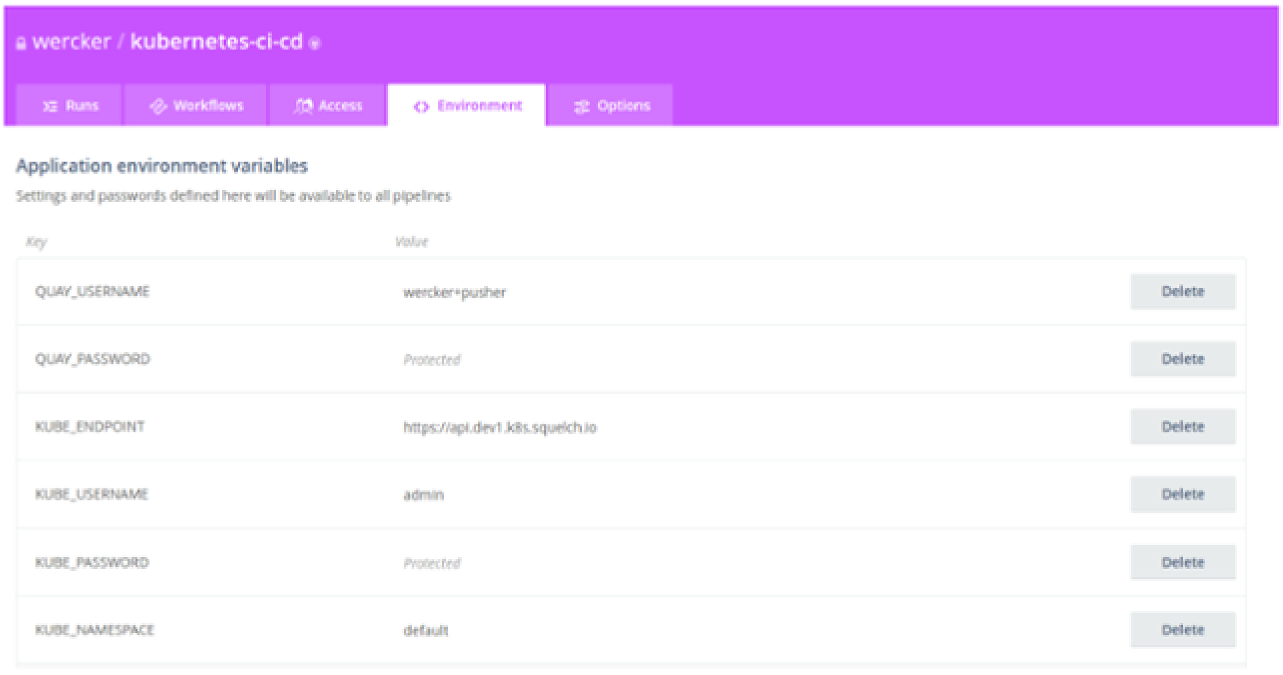
Define what starts this **pipeline** and which yml pipeline this **pipeline** maps to.

Name	deploy
YML Pipeline name	deploy
Hook type	[✓ Default] chain this Pipeline; Git push start this Pipeline on Git push

Configure Wercker.com GUI with our Kubernetes information:

- Go to the "Environment" tab.

- Add a variable with key: KUBE_ENDPOINT.
    - This is the URL to your Kubernetes API server. Remember that we gave Wercker.com's IPs access to this above.
    - The `kops` tool created us a Kubernetes configuration file when we created the cluster. This is located in: `~/.kube/config`.
    - Run the command to output the entire config:
        - cat ~/.kube/config.
    - This will be the URL of the `server` key.
- Add a variable with key: KUBE_USERNAME.
    - This is a username that Kubernetes uses for authentication.
    - You can find this in the `~/.kube/config` file.
    - There is a `username` key. This is the user. Unless you have changed it on build the user name is: admin.
- Add a variable with key: KUBE_PASSWORD.
    - This is the password associated with the username.
    - This is the `password` key associated with the username.
- Add a variable with key: KUBE_NAMESPACE.
    - Kubernetes allows you to segment off the cluster into what they call "namespaces". For this tutorial we will use the default namespace called "default".
    - The value for this is: default.



Application environment variables

Settings and passwords defined here will be available to all pipelines

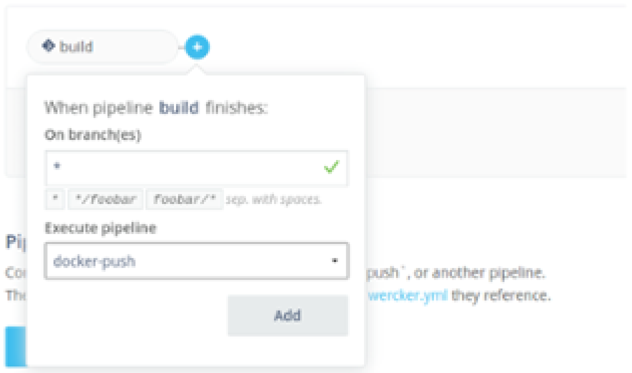
Key	Value	
QUAY_USERNAME	wercker+pusher	Delete
QUAY_PASSWORD	Protected	Delete
KUBE_ENDPOINT	https://api.dev1.k8s.squelch.io	Delete
KUBE_USERNAME	admin	Delete
KUBE_PASSWORD	Protected	Delete
KUBE_NAMESPACE	default	Delete

3) Creating the Pipeline. This strings along our "actions" above to the sequence we want which is:

build -> push docker container -> deploy to our cluster if it is the master branch.

Configure Wercker.com GUI:

- Click on the "Workflow" tab.
- Click on the "+" on the right of the "build" icon.



When pipeline build finishes:

On branch(es)

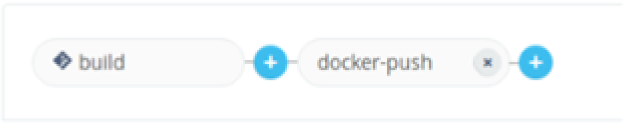
*

* */foobar foobar/* sep. with spaces.

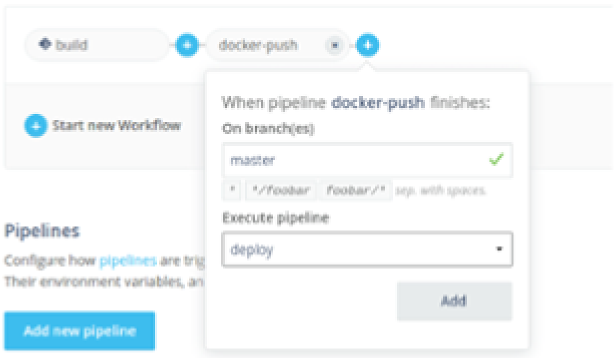
Execute pipeline

docker-push

- In the "Select pipeline" dropbox select "docker-push".
  - This tells our pipeline to run the docker-push action after a successful build

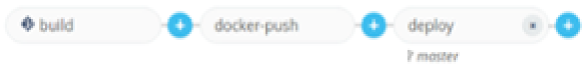


- Click on the "+" on the right of the "docker-push" action.
    - We will add the deploy next.
- This time we only want to invoke this action when it happens on the "master" branch.



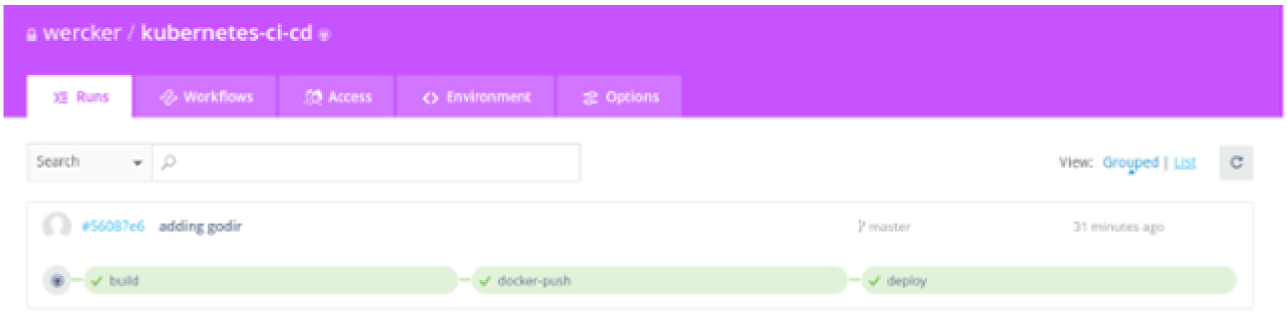
When pipeline docker-push finishes:
On branch(es): master
Execute pipeline: deploy

We have a "Workflow" that looks like this now. Notice the "master" under deploy which denotes this only happens for the "master" branch.



We have finished configuring Wercker.com to be able to deploy into our Kubernetes cluster.

To invoke it, we just have to make a small change in the Github repository and then push the code into master. You can do something as simple as changing/add something in the README.md and then pushing it in. You will get a pipeline run like this one:



You can check your Kubernetes cluster to see that this application has been deployed:

```
 $ kubectl --namespace default get pods -o wide
NAME                                        READY     STATUS    RESTARTS   AGE      IP            NO
ingress-controller-2171626210-5khg3         1/1       Running   0          13d      100.96.8.56   i
ingress-default-backend-2249275363-1vnt6    1/1       Running   0          13d      100.96.8.57
webapp-3028151003-5lb8f                     1/1       Running   0          30m      100.96.11.43  ip-
```

Check that the Kubernetes ingress has been deployed:

```
 $ kubectl  --namespace default get ing
NAME      HOSTS                             ADDRESS         PORTS     AGE
webapp    kubernetes-ci-cd-demo.wercker.com  54.172.67.240  80        49m
```

We can get the AWS ELB that Kubernetes created for us:

```
 $ kubectl  --namespace default describe svc ingress-lb
Name: ingress-lb
Namespace: default
Labels: <none>
Selector: app=ingress-controller
Type: LoadBalancer
IP: 100.69.179.81
LoadBalancer Ingress: a8338a3efc28411e6bb33123f31f6ebe-1633738224.us-east-1.elb.am
Port: http 80/TCP
NodePort: http 32660/TCP
Endpoints: 100.96.8.56:80
Port: https 443/TCP
```

```
NodePort: https 30841/TCP
Endpoints: 100.96.8.56:443
Session Affinity: None
No events.
```

We can add a DNS CNAME for our Ingress to the AWS ELB address to reach it directly. Without the CNAME we can reach it via curl and adding in a host header.

The Kubernetes ingress which is running Nginx is doing virtual host routing:

```
 $ curl -H "HOST: kubernetes-ci-cd-demo.wercker.com" a8338a3efc28411e6bb33123f31f6et
"{'cities':'San Francisco, Amsterdam, Berlin, New York','Tokyo'}"
```

Conclusion

This tutorial walked you through setting up a Kubernetes cluster in a production configuration and then we walked through the setup of wercker.com to automatically build and deploy our application.

Like Wercker?

We're hiring! Check out the careers page for open positions in Amsterdam, London and San Francisco.

As usual, if you want to stay in the loop follow us on twitter @wercker or hop on our public slack channel. If it's your first time using Wercker, be sure to tweet out your #greenbuilds, and we'll send you some swag!