
KubeCon
CloudNativeCon
North America 2018
A Basic Kubernetes Debugging Kit
curl, jq, openssl and Other Best Friends


KubeCon
CloudNativeCon
North America 2018
Who am I?
Why are we here?
What does fate have in store for us?

About Me

- In IT since my first job helping out with computers in my high school in 1994
- Past employers: Capital One, CoreOS, Red Hat, among others
- Exposed to Kubernetes in early 2015 and working with it full-time since late 2015

Currently a Solutions Architect for MESOSPHERE (we're hiring!)

Pronouns: he/him
Blood type: Caffeine-positive

Contact info:

- jthompson@mesosphere.com
- Twitter: @caffeinepresent
- Kubernetes Slack: @kensey
- LinkedIn

Why the traditional low-level tools matter

...vs. new tools:

Why the traditional low-level tools matter

...vs. new tools:

Sometimes new tools don't install seamlessly.

Why the traditional low-level tools matter

...vs. new tools:

Sometimes new tools don't install seamlessly.

Sometimes new tools break.

Why the traditional low-level tools matter

...to your daily operations

Why the traditional low-level tools matter

...to your daily operations

These tools are often already there, ready to use.

Why the traditional low-level tools matter

...to your daily operations

These tools are often already there, ready to use.

You'll get further with tools you're experienced with, all other things being equal.


Basic Tools
A quick overview

kubectl

Debugging things

kubectl attach — watch the stdout of a container in real time

kubectl cp — copy files into and out of containers (note: requires tar binary in the container)

kubectl

Debugging things

kubectl attach — watch the stdout of a container in real time

```
$ kubectl attach bash-rand-77d55b86c7-bntxs
Defaulting container name to bash-rand.
Use 'kubectl describe pod/ -n default' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
bda8caf33667184819aa0751d10fb27a
d18fa4717406dcbfe128ce717b2fb93a
4e0a7cc7c90f0192718cf6083b53b004
8039295c533b150c1a2fa60639e80588
```

kubectl cp — copy files into and out of containers (note: requires tar binary in the container)

kubectl

Debugging things

kubectl attach — watch the stdout of a container in real time

```
$ kubectl attach bash-rand-77d55b86c7-bntxs
Defaulting container name to bash-rand.
Use 'kubectl describe pod/ -n default' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
bda8caf33667184819aa0751d10fb27a
d18fa4717406dcbfe128ce717b2fb93a
4e0a7cc7c90f0192718cf6083b53b004
8039295c533b150c1a2fa60639e80588
```

kubectl cp — copy files into and out of containers (note: requires tar binary in the container)

```
$ kubectl cp default/bash-rand-77d55b86c7-bntxs:/root/rand .
tar: Removing leading `/' from member names
$ cat rand
567bea045d8b80cd6d007ced02849ac4
```

kubectl

Finding out more about what's going on than you ever wanted to know

kubectl -v — increase the verbosity of kubectl

kubectl

Finding out more about what's going on than you ever wanted to know

kubectl -v — increase the verbosity of kubectl

```
$ kubectl get nodes
NAME             STATUS   ROLES    AGE   VERSION
192.168.100.10   Ready    master   43h   v1.12.1
192.168.100.20   Ready    <none>   43h   v1.12.1
192.168.100.21   Ready    <none>   43h   v1.12.1</none></none>
```

kubectl

Finding out more about what's going on than you ever wanted to know

kubectl -v — increase the verbosity of kubectl

```
$ kubectl get nodes
NAME             STATUS   ROLES    AGE   VERSION
192.168.100.10   Ready    master   43h   v1.12.1
192.168.100.20   Ready    <none>   43h   v1.12.1
192.168.100.21   Ready    <none>   43h   v1.12.1</none></none>
```

```
$ kubectl -v 6 get nodes
I1211 11:10:28.611959   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.612482   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.614383   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.617867   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.629567   24842 round_trippers.go:405] GET https://192.168.122.138:6443/api/v1/nodes?limit=500 200 OK in 11 m
I1211 11:10:28.630279   24842 get.go:558] no kind is registered for the type v1beta1.Table in scheme "k8s.io/kubernetes/pkg
NAME             STATUS   ROLES    AGE   VERSION
[...]
```

kubectl

Finding out more about what's going on than you ever wanted to know

kubectl -v — increase the verbosity of kubectl

```
$ kubectl get nodes
NAME             STATUS   ROLES    AGE   VERSION
192.168.100.10   Ready    master   43h   v1.12.1
192.168.100.20   Ready    <none>   43h   v1.12.1
192.168.100.21   Ready    <none>   43h   v1.12.1</none></none>
```

```
$ kubectl -v 6 get nodes
I1211 11:10:28.611959   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.612482   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.614383   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.617867   24842 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:10:28.629567   24842 round_trippers.go:405] GET https://192.168.122.138:6443/api/v1/nodes?limit=500 200 OK in 11 m
I1211 11:10:28.630279   24842 get.go:558] no kind is registered for the type v1beta1.Table in scheme "k8s.io/kubernetes/pkg
NAME             STATUS   ROLES    AGE   VERSION
[...]
```

```
$ kubectl -v 99 get nodes
[...]
I1211 11:12:16.391995   25478 loader.go:359] Config loaded from file /home/kensey/bootkube/cluster/auth/kubeconfig
I1211 11:12:16.392257   25478 round_trippers.go:386] curl -k -v -XGET  -H "Accept: application/json;as=Table;v=v1beta1;g=me
I1211 11:12:16.402463   25478 round_trippers.go:405] GET https://192.168.122.138:6443/api/v1/nodes?limit=500 200 OK in 10 m
I1211 11:12:16.402483   25478 round_trippers.go:411] Response Headers:
I1211 11:12:16.402490   25478 round_trippers.go:414]     Content-Type: application/json
I1211 11:12:16.402496   25478 round_trippers.go:414]     Date: Tue, 11 Dec 2018 19:12:16 GMT
I1211 11:12:16.402572   25478 request.go:942] Response Body: {"kind":"Table",...
```

kubectl

Getting help on the fly

kubectl explain — get help with the structure of a resource

Also look at: kubectl completion, kubectl auth can-i, kubectl api-resources

kubectl

Getting help on the fly

kubectl explain — get help with the structure of a resource

```
$ kubectl explain crd
KIND:     CustomResourceDefinition
VERSION:  apiextensions.k8s.io/v1beta1

DESCRIPTION:
     CustomResourceDefinition represents a resource that should be exposed on
     the API server. Its name MUST be in the format <.spec.name>.<.spec.group>.

FIELDS:
[...]
```

Also look at: kubectl completion, kubectl auth can-i, kubectl api-resources

kubectl

Getting help on the fly

kubectl explain — get help with the structure of a resource

```
$ kubectl explain crd
KIND:     CustomResourceDefinition
VERSION:  apiextensions.k8s.io/v1beta1

DESCRIPTION:
     CustomResourceDefinition represents a resource that should be exposed on
     the API server. Its name MUST be in the format <.spec.name>.<.spec.group>.

FIELDS:
[...]
```

```
$ kubectl explain crd.spec
KIND:     CustomResourceDefinition
VERSION:  apiextensions.k8s.io/v1beta1

RESOURCE: spec <Object>

DESCRIPTION:
     Spec describes how the user wants the resources to appear

     CustomResourceDefinitionSpec describes how a user wants their resource to
     appear

FIELDS:
[...]
```

Also look at: kubectl completion, kubectl auth can-i, kubectl api-resources

kubectl

Output control

kubectl [command] -o [format] — render [command] output as [format]

Also look at: JSONPath Support

kubectl

Output control

kubectl [command] -o [format] — render [command] output as [format]

```
$ kubectl get deploy bash-rand -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2018-12-11T05:20:32Z
  generation: 1
  labels:
    run: bash-rand
[...]
```

Also look at: JSONPath Support

kubectl

Output control

kubectl [command] -o [format] — render [command] output as [format]

```
$ kubectl get deploy bash-rand -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2018-12-11T05:20:32Z
  generation: 1
  labels:
    run: bash-rand
[...]
```

```
$ kubectl get deploy bash-rand -o json
{
    "apiVersion": "extensions/v1beta1",
    "kind": "Deployment",
    "metadata": {
        "annotations": {
            "deployment.kubernetes.io/revision": "1"
        },
        "creationTimestamp": "2018-12-11T05:20:32Z",
        "generation": 1,
        "labels": {
            "run": "bash-rand"
        },
[...]
```

Also look at: JSONPath Support

curl

Adding headers to requests:

Often used for:

- setting accepted content types for replies
- setting content type of posted content
- injecting bearer auth tokens

```
curl --header "Authorization: Bearer [token]" [API server URL]
```

curl

Building and submitting requests:

- GET: request is contained in the URL itself (default method) — used to read/list/watch resources
- POST: submit a data blob to create resources
- PATCH: submit a data blob to merge-update resources
- PUT: submit a data blob to replace a resource
- DELETE: submit options to control deletion of a resource

```
$ curl --cert client.cert --key client.key --cacert cluster-ca.cert \
https://192.168.100.10:6443/api/v1/namespaces/default/pods
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/namespaces/default/pods",
    "resourceVersion": "126540"
  },
  "items": [
```

Reference examples: Pod API write operations, Pod API read operations

curl and the Kubernetes API

Watching changes:

```
$ curl --cert client.cert --key client.key --cacert cluster-ca.cert \
https://192.168.100.10:6443/api/v1/namespaces/default/pods?watch=true
{"type":"ADDED","object":{"kind":"Pod","apiVersion":"v1","metadata":{"name":"bash-
$ curl --cert client.cert --key client.key --cacert cluster-ca.cert \
https://192.168.100.10:6443/api/v1/nodes?watch=true
{"type":"ADDED","object":{"kind":"Node","apiVersion":"v1","metadata":{"name":"192.
{"type":"ADDED","object":{"kind":"Node","apiVersion":"v1","metadata":{"name":"192.
{"type":"ADDED","object":{"kind":"Node","apiVersion":"v1","metadata":{"name":"192.
{"type":"MODIFIED","object":{"kind":"Node","apiVersion":"v1","metadata":{"name":"1
```

curl and the Kubernetes API

Feeding results to other things:

Basic method — bash while read loop

Something completely different — bash coproc

curl and the Kubernetes API

Feeding results to other things:

Basic method — bash while read loop

```
$ while read -r serverevent; do echo "$serverevent" \
| jq '.["operation"] = .type | .["node"] = .object.metadata.name | { "operation": .operation, "node": .node}' ; \
done < <(curl -sN --cert client.cert --key client.key --cacert cluster-ca.cert \
https://192.168.100.10:6443/api/v1/nodes?watch=true)
{
  "operation": "ADDED",
  "node": "192.168.100.10"
}
[...]
```

Something completely different — bash coproc

curl and the Kubernetes API

Feeding results to other things:

Basic method — bash while read loop

```
$ while read -r serverevent; do echo "$serverevent" \
| jq '.["operation"] = .type | .["node"] = .object.metadata.name | { "operation": .operation, "node": .node}' ; \
done < <(curl -sN --cert client.cert --key client.key --cacert cluster-ca.cert \
https://192.168.100.10:6443/api/v1/nodes?watch=true)
{
  "operation": "ADDED",
  "node": "192.168.100.10"
}
[...]
```

Something completely different — bash coproc

```
#!/bin/bash

coproc curl -sN --cacert cluster-ca.cert --cert ./client.cert --key ./client.key \
https://192.168.100.10:6443/api/v1/nodes?watch=true

exec 5<&${COPROC[0]}

while read -ru 5 serverevent; do
  if [[ $(echo $serverevent | jq -r '.type') == "ADDED" ]]; then
    echo "Added node $(echo $serverevent | jq -r '.object.metadata.name') in namespace \
$(echo $serverevent | jq '.object.metadata.namespace')"
  fi
done

trap 'kill -TERM $COPROC_PID' TERM INT
```

openssl

Working with certificate trust chains:

Some certificates verify...

...and some don't.

openssl

Working with certificate trust chains:

Some certificates verify...

```
$ openssl verify -CAfile cluster-ca.cert client.cert
client.cert: OK
```

...and some don't.

openssl

Working with certificate trust chains:

Some certificates verify...

```
$ openssl verify -CAfile cluster-ca.cert client.cert
client.cert: OK
```

...and some don't.

```
$ openssl verify client.cert
O = system:masters, CN = admin
error 20 at 0 depth lookup: unable to get local issuer certificate
error client.cert: verification failed
```

openssl

Working with a live server:

```
$ openssl s_client -connect 192.168.100.10:6443
CONNECTED(00000004)
depth=0 O = kube-master, CN = kube-apiserver
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 O = kube-master, CN = kube-apiserver
verify error:num=21:unable to verify the first certificate
verify return:1
---
Certificate chain
 0 s:O = kube-master, CN = kube-apiserver
   i:O = efc441d8-e199-4031-a2c9-f2e0433cd550, OU = bootkube, CN = kube-ca
---
Server certificate
-----BEGIN CERTIFICATE-----
[...]
```

Other useful basic tools:

dig, tcpdump, netcat and ss

- No detailed examples here (because I don't have anything earth-shattering to show)
- Remember that network interfaces are different inside the container (we'll see this later)
- Be aware of dependencies
  - DNS works differently in musl-based containers like Alpine
- Pay attention to tools being deprecated
  - ifconfig/route -> ip addr show, ip route show
  - netstat -> ss

jq

Basic usage:

```
[some JSON content] | jq [flags] [filter expression]
```

jq

Easy filters:

The dot filter — prettyprints input unchanged... asterisk

Using the hierarchy to filter the input

jq

Easy filters:

The dot filter — prettyprints input unchanged... asterisk

```
$ echo '{ "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }' | \
jq .
{
  "key": "value",
  "key2": {
    "subkey": "value",
    "subkey2": "value2"
  },
  "key3": 12,
  "key4": [
    "one",
    "two"
  ]
}
```

Using the hierarchy to filter the input

jq

Easy filters:

The dot filter — prettyprints input unchanged... asterisk

```
$ echo '{ "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }' | \
jq .
{
  "key": "value",
  "key2": {
    "subkey": "value",
    "subkey2": "value2"
  },
  "key3": 12,
  "key4": [
    "one",
    "two"
  ]
}
```

Using the hierarchy to filter the input

```
$ echo '{ "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }' | \
jq .key2
{
  "subkey": "value",
  "subkey2": "value2"
}
```

jq

Easy filters:

The dot filter — prettyprints input unchanged... asterisk

```
$ echo '{ "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }' | \
jq .
{
  "key": "value",
  "key2": {
    "subkey": "value",
    "subkey2": "value2"
  },
  "key3": 12,
  "key4": [
    "one",
    "two"
  ]
}
```

Using the hierarchy to filter the input

```
$ echo '{ "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }' | \
jq .key2
{
  "subkey": "value",
  "subkey2": "value2"
}
```

```
$ echo '{ "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }' | \
jq .key4[1]
"two"
```

jq

More advanced — functions

- select: cherry-pick a piece of the input by criteria
- contains: match a value that contains an element

```
$ echo '[ { "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }, { "
jq '.[] | select( .key2.subkey | contains ( "value3" ) )'
{
  "key": "value",
  "key2": {
    "subkey": "value3",
    "subkey2": "value4"
  },
  "key3": 12,
  "key4": [
    "three",
    "four"
  ]
}
```

Also check out: the jq cookbook and the jq manual

jq

Scripting beyond one-liners:

jq allows creating a script file for readability (also great for easier source control)

```
$ cat script.jq
.[]
| select( .key2.subkey
| contains ( "value3" ) )

$ echo '[ { "key": "value", "key2": { "subkey": "value", "subkey2": "value2" }, "key3": 12, "key4": [ "one", "two" ] }, { "
jq -f script.jq
{
  "key": "value",
  "key2": {
    "subkey": "value3",
    "subkey2": "value4"
  },
  "key3": 12,
  "key4": [
    "three",
    "four"
  ]
}
```

Your bash prompt

Keeping out of trouble:

Just like the $ changing to a # when you're root, a visual reminder of where you're working and who you are can be useful.

Your bash prompt

Setting the prompt in your .bash_profile:

Now, when we have $KUBECONFIG set:

Your bash prompt

Setting the prompt in your .bash_profile:

```
prompt_set() {
if [ "$KUBECONFIG" != "" ]; then
  PROMPT_KUBECONTEXT="k8s:$(kubectl config current-context 2>/dev/null)\n"
fi
PS1="${PROMPT_KUBECONTEXT}[\u@\h \W]\$ "
}
```

Now, when we have $KUBECONFIG set:

Your bash prompt

Setting the prompt in your .bash_profile:

```
prompt_set() {
if [ "$KUBECONFIG" != "" ]; then
  PROMPT_KUBECONTEXT="k8s:$(kubectl config current-context 2>/dev/null)\n"
fi
PS1="${PROMPT_KUBECONTEXT}[\u@\h \W]\$ "
}
```

Now, when we have $KUBECONFIG set:

```
k8s:admin@local
[kensey@sleeper-service ~]$
```


Getting tools where you need them

Plugging system tools into a running container

...with Docker:

```
$ docker ps | grep bash-rand
77c73df2f584        fedora                "/bin/bash -c 'while…"   6 hours ago         Up 6 hours
cfb38a9eca9e        k8s.gcr.io/pause:3.1   "/pause"                 6 hours ago         Up 6 hours
core@coreos-k8s-worker1 ~ $ docker run -it --net=container:77c73df2f584 fedora /bin/bash
[root@bash-rand-77d55b86c7-bntxs /]# yum install iproute
[...]
Installed:
  iproute-4.18.0-3.fc29.x86_64        iproute-tc-4.18.0-3.fc29.x86_64
  libmnl-1.0.4-8.fc29.x86_64          linux-atm-libs-2.5.1-21.fc29.x86_64

Complete!
[root@bash-rand-77d55b86c7-bntxs /]# ip a
1: lo: <loopback,up,lower_up> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if8: <broadcast,multicast,up,lower_up> mtu 1450 qdisc noqueue state UP group default
    link/ether 0a:58:0a:02:02:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.2.2.3/24 scope global eth0
       valid_lft forever preferred_lft forever</broadcast,multicast,up,lower_up></loopback,up,lower_up>
```

Plugging system tools into a running container

...with nsenter — Getting the right PID:

```
$ kubectl get po -o wide
NAME                          READY   STATUS    RESTARTS   AGE     IP          NODE              NOMINATED NODE
bash-rand-77d55b86c7-bntxs    1/1     Running   0          5h48m   10.2.2.3    192.168.100.21    <none>
```

```
$ ps -A -o pid,ppid,netns,cmd | grep bash
  804   803 4026531993 -bash
 2484  2465          - /bin/bash -c while true; do openssl rand -hex 16 | tee /root/rand; sleep 3; done
 6218   804 4026531993 grep --colour=auto bash
```

Plugging system tools into a running container

...with nsenter — Running in the namespace:

```
$ sudo nsenter -t 2484 -n
Update Strategy: No Reboots
coreos-k8s-worker1 core # ip a
1: lo: <loopback,up,lower_up> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if8: <broadcast,multicast,up,lower_up> mtu 1450 qdisc noqueue state UP group default
    link/ether 0a:58:0a:02:02:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.2.2.3/24 scope global eth0
       valid_lft forever preferred_lft forever</broadcast,multicast,up,lower_up></loopback,up,lower_up>
```

Running a debug container

Which one should you run?

Running a debug container

Which one should you run?

It's really up to you. The important thing is that it have all the tools you need in it — whether it's a standard distro container or one you build yourself.

Running a debug container

Which one should you run?

It's really up to you. The important thing is that it have all the tools you need in it — whether it's a standard distro container or one you build yourself.

(That said, if you aren't already building one yourself, do it occasionally, so that you have a sense of the delta between what you need and what you're using.)

Running a debug container

What should be in it?

- Standard system tools
- Filtering and parsing tools
- Tools for debugging and managing your apps


KubeCon
CloudNativeCon
North America 2018
Demos

Final thoughts

Remember:

- Don't panic
- Know your tools
- Know what lies *under* your tools
- It's not really new — it's just Linux


KubeCon
CloudNativeCon
North America 2018
Questions?
(Thank you!)
Link to slides: