

Overview: Infrastructure as Code to support automation and workload migration Part #2D

Note: Template background, notes, references, identification and other details, and some content, have been removed from this version.

Jeff Barnes

Feb 10, 2018 (v 0.4)

For information only. We acknowledge incomplete attribution after the notes were stripped from his version

Agenda

Definition - Cloud Native

1. Single-stage versus Multi-stage builds in Docker

- Single-Stage Build (Vulnerability Scan #1)
- Single-Stage Build (Vulnerability Scan #2)
- Multi-Stage Builds
- Monitor and Protect

2. Application Deployment

- Application Deployment
- Local deploy using Kubernetes
- Kubernetes on other platforms
- Containers on Public Clouds

Questions/Additional Slides

Cloud Native

A cloud-native application is a distributed, elastic and horizontal scalable system composed of (micro)services which isolates state in a minimum of stateful components. The application and each self-contained deployment unit of that application is designed according to cloud-focused design patterns and operated on a self-service elastic platform.

Even though the specific tools and patterns may differ, cloud-native organizations and applications follow a fairly consistent pattern

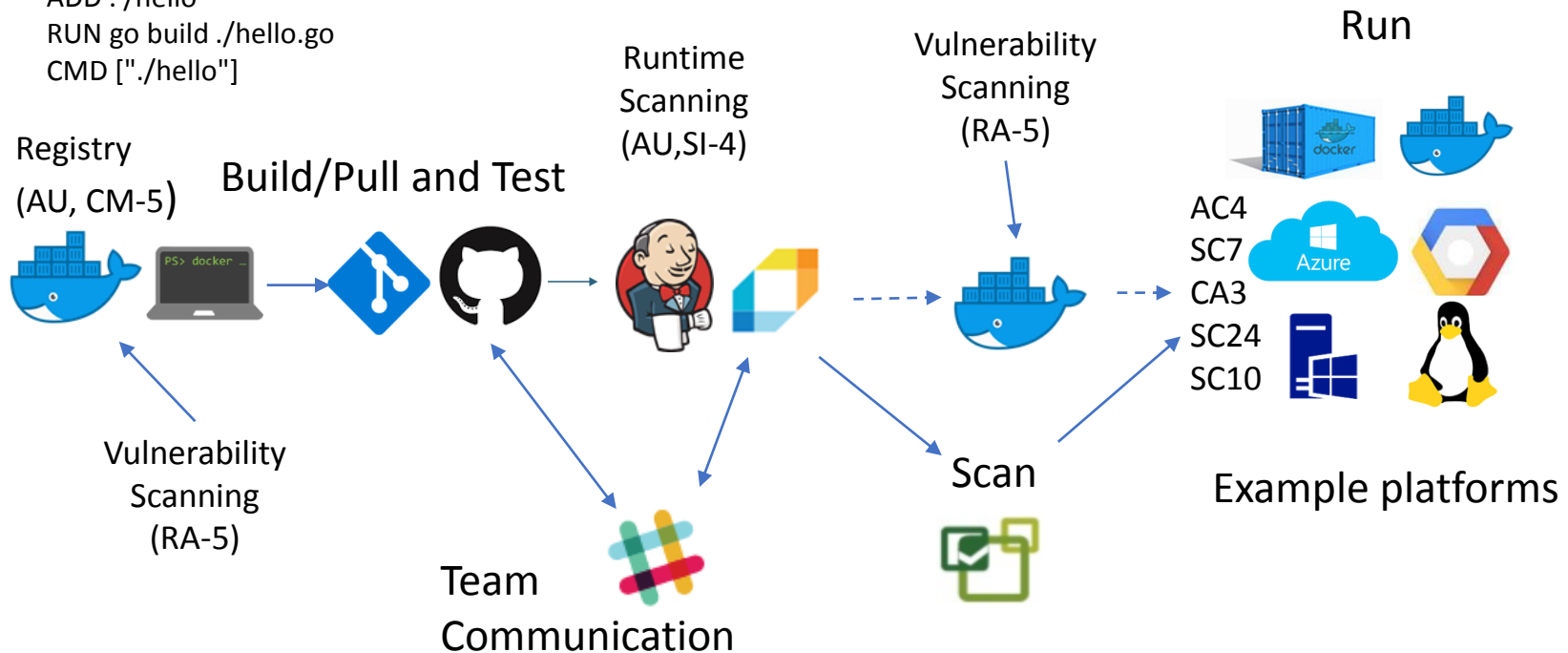
Kratzke, N., Quint, P.C.: Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study. Journal of Systems and Software 126(April), 1–16 (2017)

Single-Stage Build

Dockerfile

```
FROM golang:alpine
RUN mkdir -p /hello
WORKDIR /hello
ADD . /hello
RUN go build ./hello.go
CMD ["/hello"]
```


Docker Single-stage build, includes CI/D tools Git, GitHub, Slack, Jenkins, Docker Hub and deployment stack





Container Management

Single-Stage Build

```
\GitHub\go [master = +0 ~2 -0 !]> git add hello.go
\GitHub\go [master = +0 ~1 -0 | +0 ~1 -0 !]> git commit
```

 **github** APP 4:11 PM
[go:master] 1 new commit by Jeff Barnes:
[d01aa2c] Delete Dockerfile - Jeff Barnes
[go:master] 1 new commit by Jeff Barnes:
[6278b1d] Add files via upload - Jeff Barnes



 **jenkins** APP 4:15 PM
DevJob One - #49 Success after 1 min 18 sec ([Open](#))

 **jenkins** APP 4:29 PM
DevJob One - #50 Success after 1 min 19 sec ([Open](#))

Team update sent to Slack Chanel. Can also do versioning control and add a Webhook to automatically trigger Jenkins build on Github update.

Build runs in Jenkins, can include QA, testing, Blue/Green deployment, etc. In this example, compiles GO binary, builds and runs in Docker containers

[add description](#)

S	W	Name ↓	Last Success	Last Failure	Last Duration
		DevJob One	3 days 23 hr - #52	N/A	14 sec

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Manually verify/validate images using Neuvector against CIS database and other controls. Can also be automated in the build pipeline

Contains CIS vulnerabilities! Automated verification would fail the build in Jenkins

Containers		Nodes					
Name	OS	Node	Image	Status	High	Me...	Time
neuvevector.enfor...		otg-ubuntu-op...	neuvevector/enf...		0	0	
pensive_bell	debian:9	otg-ubuntu-op...	hello	Finished	6	11	just now

Vulnerability Scan #1

Containers		Nodes					
Name	OS	Node	Image	Status	High	Me...	Time
neuvector.enfor...		otg-ubuntu-op...	neuvector/enf...		0	0	
pensive_bell	debian:9	otg-ubuntu-op...	hello	Finished	6	11	just now

Single-stage build includes libraries and OS, which contain vulnerabilities!

Name	Urgency	Package	Version
CVE-2017-8804	High	glibc	2.24-11+deb9u1
CVE-2017-8831	High	linux	4.9.51-1
CVE-2013-7445	High	linux	4.9.51-1
CVE-2017-7487	High	linux	4.9.51-1
CVE-2017-8890	High	linux	4.9.51-1
CVE-2016-2779	High	util-linux	2.29.2-1
CVE-2017-9038	Medium	binutils	2.28-5
CVE-2017-9043	Medium	binutils	2.28-5
CVE-2017-9040	Medium	binutils	2.28-5
CVE-2017-9042	Medium	binutils	2.28-5

Single-stage Dockerfile

```
FROM golang
RUN mkdir -p /hello
WORKDIR /hello
ADD . /hello
RUN go build ./hello.go
CMD ["/hello"]
```

Post build scan using Neuvector Enforcer Container Security Broker
Can also run docker-bench

Plus the build size is 735 MB for a Hello World program

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello	v2	aa9b89ac96cf	2 minutes ago	735MB
hello	v0	17cbe44ed49a	10 minutes ago	5.86MB

Vulnerability Scan #2 (Same code)

The screenshot shows the Jenkins Aqua Security Scanner interface. The top navigation bar includes 'Jenkins', 'DevJobOne2', '#5', and 'Aqua Security Scanner'. A sidebar on the left contains links like 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete Build', 'Open Blue Ocean', 'Environment Variables', 'Git Build Data', 'No Tags', 'Aqua Security Scanner', and 'Previous Build'. The main content area displays a 'Vulnerability Report: hello:v2' from the 'Local Docker Engine'. It shows a score of 0 (High) out of 1 (Medium), 1 (Low), 12 (Negl.), and 1.8 (Score Avg.). Below this, it states 'The following vulnerabilities were found:' and lists 15 items with columns for Name, File, Severity, Score, and Publish Date. The vulnerabilities include CVE-2005-2541 (tar), CVE-2016-2779 (util-linux), CVE-2007-6755 (openssl), WS-2017-0195 (/usr/local/go/misc/tour/static/lib/jquery.min.js), CVE-2017-10790 (libtasn1-6), CVE-2017-3735 (openssl), CVE-2013-7040 (python2.7), CVE-2010-0928 (openssl), CVE-2011-2207 (dirmngr), CVE-2011-3374 (apt), CVE-2011-4116 (perl), CVE-2012-3878 (perl), CVE-2017-15298 (git), DSA-4007-1 (curl), and DSA-4008-1 (wget).

Name	File	Severity	Score	Publish Date
CVE-2005-2541	tar	negligible	10	2005-08-10
CVE-2016-2779	util-linux	negligible	7.2	2017-02-07
CVE-2007-6755	openssl	negligible	5.8	2013-10-11
WS-2017-0195	/usr/local/go/misc/tour/static/lib/jquery.min.js	medium	5.3	
CVE-2017-10790	libtasn1-6	negligible	5	2017-07-01
CVE-2017-3735	openssl	negligible	5	2017-08-28
CVE-2013-7040	python2.7	negligible	4.3	2014-05-19
CVE-2010-0928	openssl	negligible	4	2010-03-05
CVE-2011-2207	dirmngr	negligible	0	
CVE-2011-3374	apt	negligible	0	
CVE-2011-4116	perl	negligible	0	
CVE-2012-3878	perl	negligible	0	
CVE-2017-15298	git	negligible	0	2017-10-14
DSA-4007-1	curl		0	
DSA-4008-1	wget	low	0	

Single-stage build includes libraries and OS, which contain vulnerabilities!
Scan occurs during build, can log vulnerabilities or fail build

Single-stage Dockerfile

```
FROM golang
RUN mkdir -p /hello
WORKDIR /hello
ADD . /hello
RUN go build ./hello.go
CMD ["/hello"]
```

Plus the build size is 735 MB for a Hello World program

Vulnerability Scan during build with Aqua Container Security Broker, log or fail build on vulnerabilities

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello	v2	aa9b89ac96cf	2 minutes ago	735MB
hello	v0	17cbe44ed49a	10 minutes ago	5.8GMB

Multi-Stage build (Same code)

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello	v2	aa9b89ac96cf	2 minutes ago	735MB
hello	v0	17cbe44ed49a	10 minutes ago	5.86MB

Multi-stage build contains fewer libraries and OS, minimal vulnerabilities

Program contains minimal vulnerability (inherently more secure) plus only 5.8 MB (versus 735 MB) in size

The screenshot shows the Jenkins Aqua Security Scanner interface. The main heading is 'Vulnerability Report: hello:v0'. Below it, it says 'From Local Docker Engine'. A score is displayed as '0 0 1 0 0.0' with a legend 'High Medium Low Negl. Score Avg.'. A message states 'The following vulnerabilities were found:'. A table lists one vulnerability: CVE-2017-15650 in the musl file, with a low severity score of 0, published on 2017-10-19. A note at the bottom says 'This scan report was generated by Aqua Container Security v2.5.3 on 2017-10-30, 08:56:34. Aqua'. The left sidebar contains navigation links like 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete Build', 'Open Blue Ocean', 'Environment Variables', 'Git Build Data', 'No Tags', 'Aqua Security Scanner', and 'Previous Build'.


Multi-Stage Dockerfile

```
FROM golang AS build-env
RUN mkdir -p /app
WORKDIR /app
ADD . /app
RUN go build ./hello.go
```

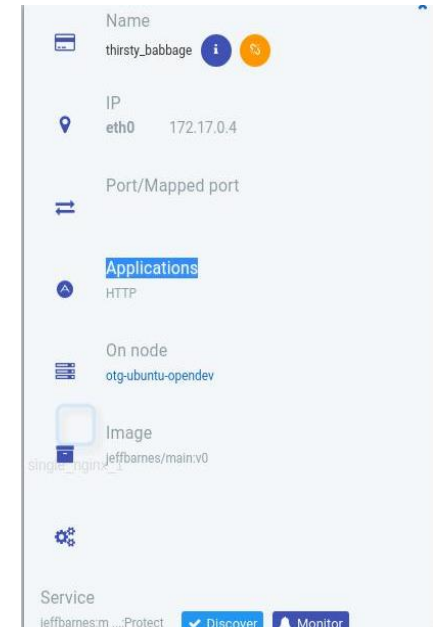
```
# final stage
FROM alpine
WORKDIR /app
COPY --from=build-env /app /app
RUN cd /app
CMD ./hello
```


Monitor and Protect Code once in Containers

Container Security Broker scans, monitors and can protect running containers on host/platform

Application	Client	Server	Client bytes	Server byt...	Rule	Action	Duration
HTTP		850i	140 bytes	74 bytes	0	Open	00:00:14
		850i	140 bytes	74 bytes	0	Open	00:00:14
		850i	140 bytes	74 bytes	0	Open	00:00:14
		850i	3.3 KB	1.3 KB	0	Open	00:00:14

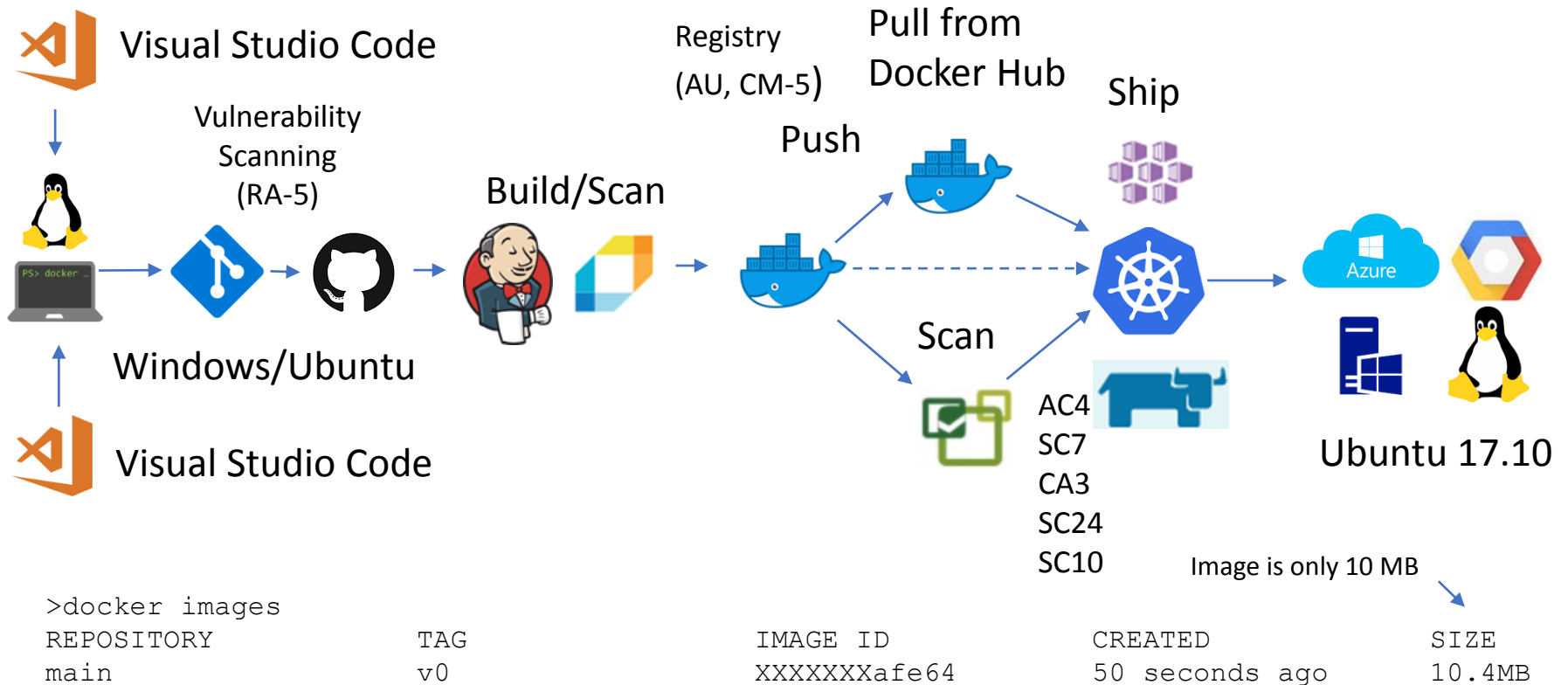
Similar services available from CSP, both managed and DIY



The screenshot shows the CSB interface for a service named 'thirsty_babbage'. It displays the IP address '172.17.0.4' and the port 'eth0'. Under the 'Applications' section, 'HTTP' is listed. The 'On node' section shows 'otg-ubuntu-opendev'. The 'Image' section shows 'jeffbarnes/main:v0'. At the bottom, there are buttons for 'Discover' and 'Monitor'.

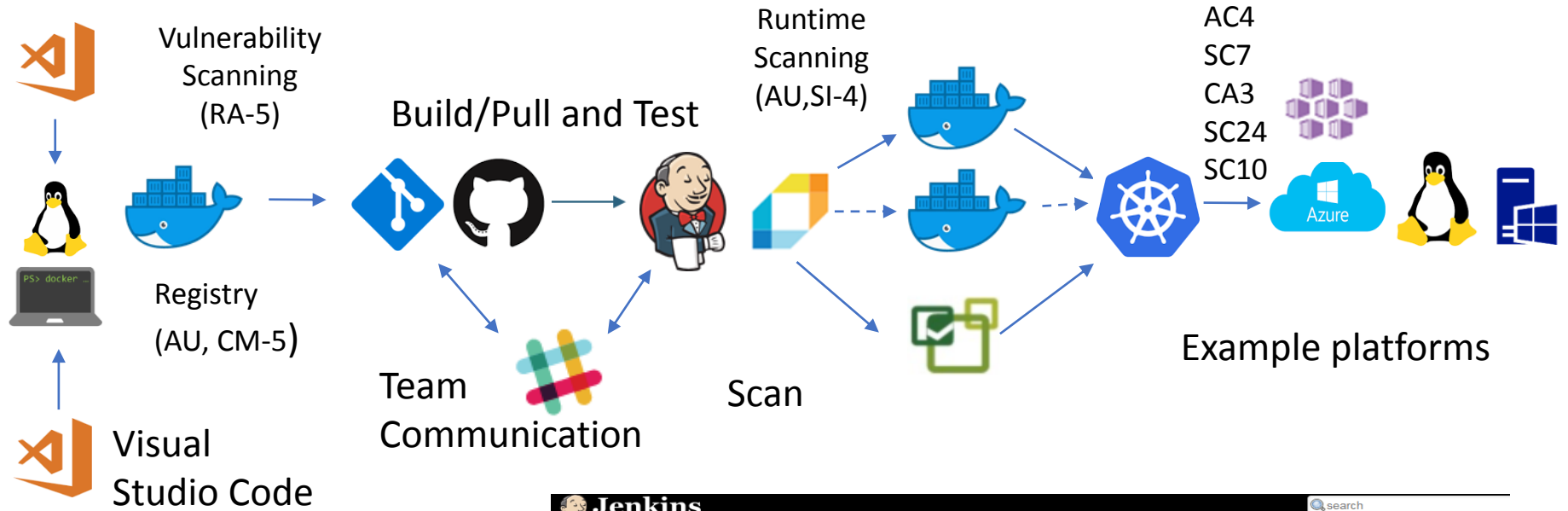
Application maps, container security

#2 Application Deployment



Note: example dev build deploys Kubernetes cluster on a local host. Automate testing and deployment to public cloud using the same patterns. Deploy through CI/CD, such as Jenkins, to cloud service providers or deploy over multiple clouds

Multi-Stage build



The screenshot shows the Jenkins Aqua Security Scanner interface. The top navigation bar includes the Jenkins logo and the path: **Jenkins** > **SimpleWebServer** > **#7** > **Aqua Security Scanner**. The left sidebar contains a list of actions: **Back to Project**, **Status**, **Changes**, **Console Output**, **Edit Build Information**, **Delete Build**, **Environment Variables**, **Git Build Data**, **No Tags**, and **Aqua Security Scanner**. The main content area displays a **Vulnerability Report: main:v0** from the **Local Docker Engine**. The report shows a score of 0.0 and lists the following vulnerabilities found:

Name	File	Severity	Score	Publish Date
CVE-2017-15650	musl	low	0	2017-10-19

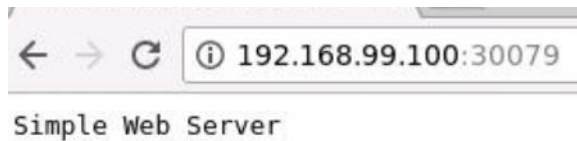
This scan report was generated by Aqua Container Security v2.5.3 on 2017-10-27, 12:08:11. Aqua

Local deploy using Kubernetes

```
kubectl run go-main --image=jeffbarnes/main:v0 --port=8500  
kubectl expose deployment go-main --type="LoadBalancer"
```

```
minikube service go-main --url  
http://192.168.99.100:30079
```

Application/service deployed and available



Scale deployment to 5 manually

```
kubectl scale deployment go-main --replicas=5  
deployment "go-main" scaled
```

✓ go-main-1774950723-8drth	minikube	Running	0	7 minutes
✓ go-main-1774950723-m5094	minikube	Running	0	7 minutes
✓ go-main-1774950723-pt19n	minikube	Running	0	7 minutes
✓ go-main-1774950723-wdlkk	minikube	Running	0	7 minutes
✓ go-main-1774950723-g5mlx	minikube	Running	0	16 minutes

Local deploy to Kubernetes

via YAML

```
kubectl apply -f deploy.yaml
```

```
! deploy.yaml x
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: main
5  spec:
6    selector:
7      matchLabels:
8        app: main
9    replicas: 3
10   template:
11     metadata:
12       labels:
13         app: main
14     spec:
15       containers:
16       - name: main
17         image: jeffbarnes/main:v0
18         ports:
19         - containerPort: 8500
20         securityContext:
21           readOnlyRootFilesystem: true
22           allowPrivilegeEscalation: false
23   ---
24   apiVersion: v1
25   kind: Service
26   metadata:
27     name: main
28     labels:
29       app: main
30   spec:
31     type: LoadBalancer
32     selector:
33       app: main
34     ports:
35     - protocol: TCP
36       port: 8500
37       targetPort: 8500
38     name: http
```

via Helm

```
helm create main
```

```
$ tree main
```

```
main
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   └── service.yaml
└── values.yaml
```

```
$ helm install --name main /home/(user)/main
```

Local deploy using Kubernetes

Now let's delete a pod and see what happens

```
kubectl delete pods go-main-1774950723-m5094
```

Kubernetes automatically re-adds it, can also provision by resources

Deployments

Name	Labels	Pods	Age	Images
go-main	run: go-main	4 / 5	2 days	jeffbarnes/main:v0

Pods

Name	Node	Status	Restarts	Age
go-main-1774950723-dw9qh	minikube	Waiting: ContainerCreating	0	3 seconds
go-main-1774950723-b55mq	minikube	Running	0	a minute
go-main-1774950723-g4494	minikube	Running	0	a minute
go-main-1774950723-gg94x	minikube	Running	0	a minute
go-main-1774950723-g5mlx	minikube	Running	0	2 days

Name	Labels	Pods	Age	Images
go-main	run: go-main	5 / 5	2 days	jeffbarnes/main:v0

Replicas restored



Local deploy using Kubernetes

Pods – scale deployment manually or automatically. Scale from 5 to 2

```
kubectl scale deployment go-main --replicas=2
```

✓ go-main-1774950723-m5094	minikube	Running	0
✓ go-main-1774950723-g5mlx	minikube	Running	0

Replica Sets

✓ go-main-1774950723	pod-template-hash: 1774950723-g5mlx	2 / 2	30 minutes	jeffbarnes/main:v0
	run: go-main			

Upgrading the Deployment: various ways to do Rolling Updates/Rollouts

```
kubectl set image deployment/hello-node hello-node=hello-node:v2
```

✓ hello-node-1038538626	pod-template-hash: 1038538626-1038538626	0 / 0	3 hours	hello-node:v1
	run: hello-node			
✓ hello-node-2387323937	pod-template-hash: 2387323937-2387323937	2 / 2	13 minutes	hello-node:v2
	run: hello-node			

```
kubectl rolling-update hello-node --image=hello-node:v2
```

```
kubectl rolling-update hello-node --rollback
```

Containers on Public Clouds

Deploy image to Azure Container Instance via Jenkins

15 seconds from
a build in Jenkins
to publically
accessible
application in Azure



Azure Container Service

```
az acs create --name XXX --dns-prefix XXX --resource-group XXX --orchestrator-  
type kubernetes --generate-ssh-keys \
```

Or any number of other private, public or hybrid cloud services: GKE,
AKS, EKS, IBM Cloud

Kubernetes Controls Matrix (WIP)

Mapping NIST SP800 53 controls to Kubernetes via the CIS Benchmark
Automate checking via Kube-bench or CASB

	A	B	C	D
1	NIST 800-53 rev4	Critical Security Control	Kubernetes Master (APIServer)	Kubernetes Worker (Kublet)
	CA-7: Continuous Monitoring SC-39: Process Isolation SC-44: Detonation Chambers SI-3: Malicious Code Protection SI-4: Information System Monitoring	Critical Security Control #8: Malware Defenses	Ensure that the admission control policy is set to EventRateLimit	
9	SI-8: Spam Protection			
	AC-4: Information Flow Enforcement CA-7: Continuous Monitoring CA-9: Internal System Connections CM-2: Baseline Configuration CM-6: Configuration Settings CM-8: Information System Component Inventory SC-20: Secure Name /Address Resolution Service (Authoritative Source) SC-21: Secure Name /Address Resolution Service (Recursive or Caching Resolver) SC-22: Architecture and Provisioning for Name/Address Resolution Service SC-41: Port and I/O Device Access	Critical Security Control #9: Limitation and Control of Network Ports	Ensure --insecure-bind-address argument is not set Ensure --insecure-port argument is set to 0 Ensure --authorization-mode argument is not set to AlwaysAllow Ensure that the --authorization-mode argument is set to Node Ensure that the --etcd-certfile and --etcd-keyfile arguments are set as appropriate Ensure that the --max-wals argument is set to 0 Ensure that a unique Certificate Authority is used for etcd	Ensure that the --read-only-port argument is set to 0 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 Ensure that the --make-iptables-util-chains argument is set to true
10	SI-4: Information System Monitoring			

`$/kube-bench master`

Run `$/kube-bench` against master or nodes to validate Kubernetes install against CIS Benchmark

```

ice) and configure the startup parameter for --wal-dir and set it to "${ETCD_WAL
DIR}"
1.5.8 Edit the etcd environment file (for example, /etc/etcd/etcd.conf) on the e
tcd server node and set the ETCD_MAX_WALS parameter to 0. Edit the etcd startu
p file (for example, /etc/systemd/system/multiuser.target.wants/etcd.service)
and configure the startup parameter for --max-wals and set it to "${ETCD_MAX_W
ALS)".
1.5.9 Follow the etcd documentation and create a dedicated certificate authority
setup for the etcd service.

-- Summary --
checks PASS
checks FAIL
checks WARN

~$

```

Basic Hygiene

- Use a non-root user inside containers

Dockerfile

```
RUN groupadd -r nodejs
RUN useradd -m -r -g nodejs nodejs
USER nodejs
```

YAML File

```
securityContext:
  runAsNonRoot: true
```

- Make the filesystem read-only

YAML File

```
securityContext:
  readOnlyRootFilesystem: true
```

- Use the “record” option for easier rollbacks
- Use Labels, Tags, (Not LATEST)
- RBAC, ABAC, Namespace, Sidecars