RSAConference2019
San Francisco | March 4–8 | Moscone Center

BETTER.

SESSION ID: HT-W02

Hacking and Hardening Kubernetes

Jay Beale

CTO
InGuardians, Inc
@jaybeale and @inguardians

Adam Crompton

Senior Security Analyst
InGuardians, Inc.
@3nc0d3r and @inguardians

Table of Contents

- Describe what Kubernetes is and does
- Demonstrate attacks on Kubernetes clusters
- Demonstrate defenses for Kubernetes clusters and workloads
- Introduce further defenses for Kubernetes
- Introduce a new Open Source tool: Peirates

What Does Kubernetes Do?

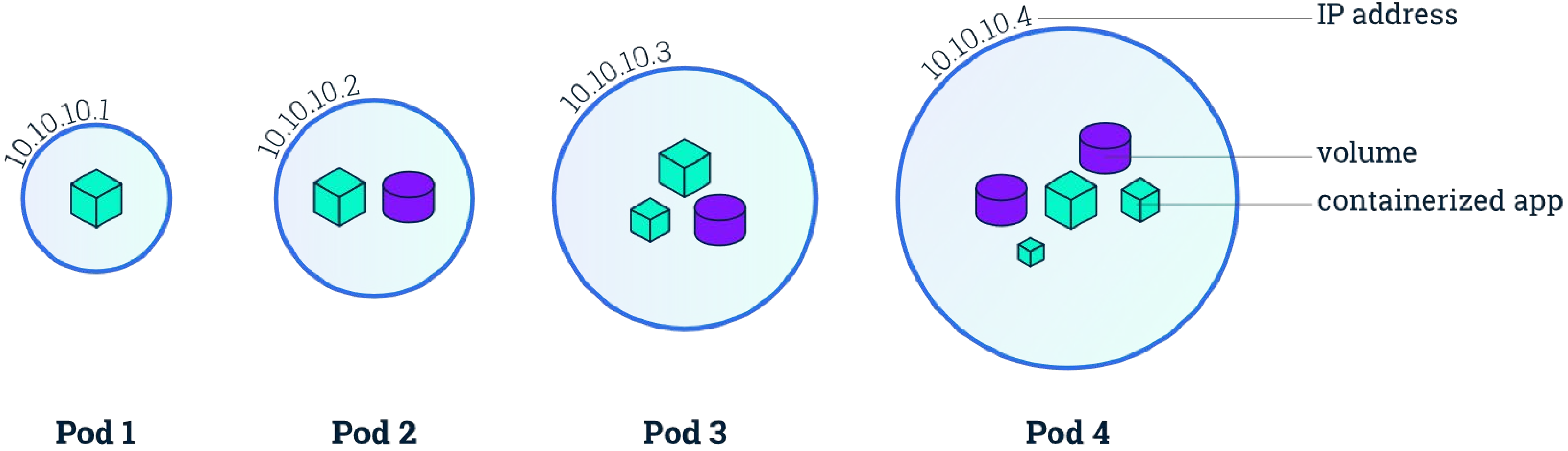
- Container orchestration
- Horizontal scaling
- Self-healing
- Automatic binpacking
- Automated rollouts and rollback
- Service Discovery and Load Balancing
- Secret and configuration management

Software-defined
Datacenter via
Container Orchestration

Kubernetes Components

- Pods
- Volumes
- Nodes
- Namespaces

Kubernetes Concept: Pods



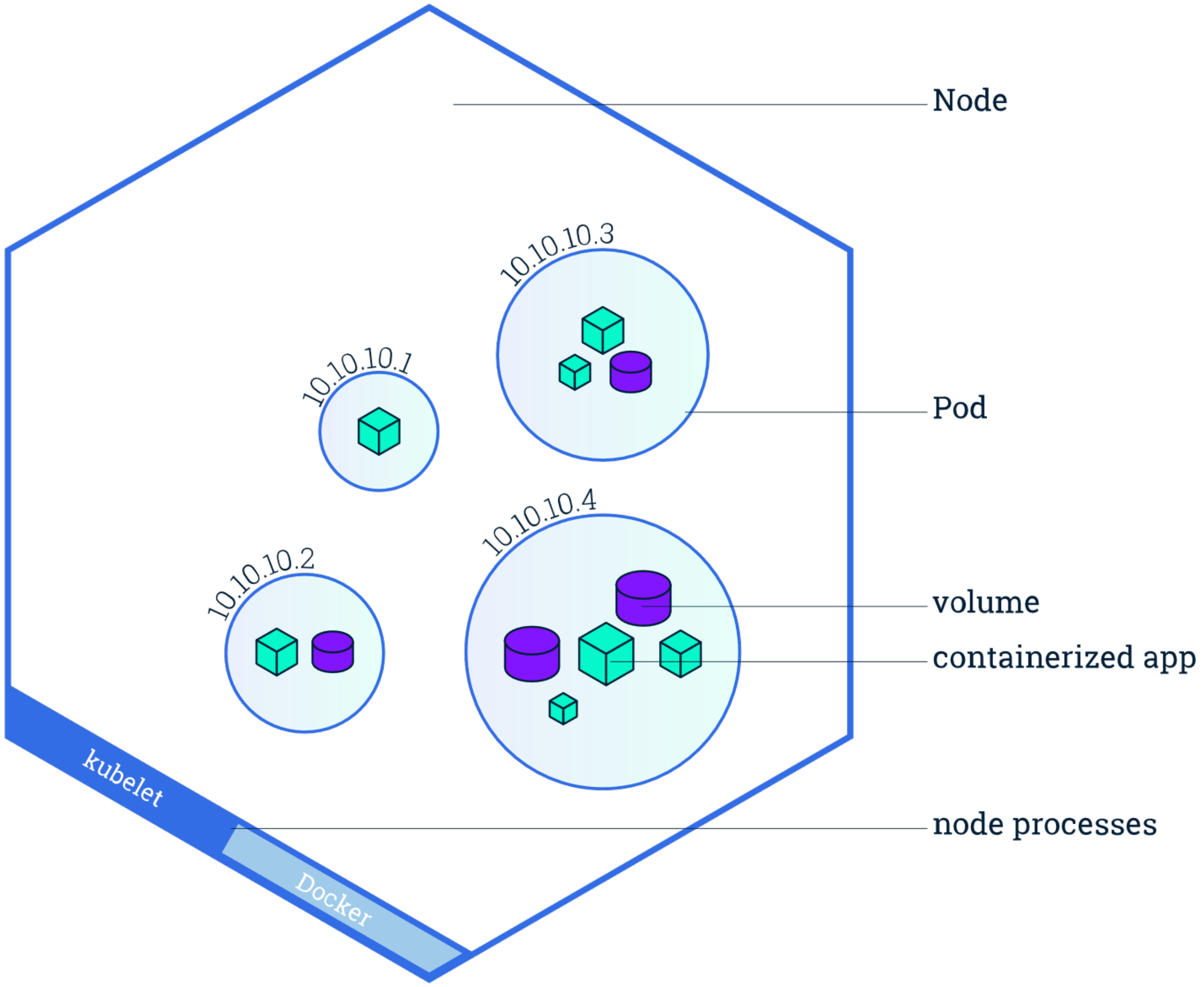
PODS ARE THE SMALLEST UNIT OF WORK IN KUBERNETES

All containers in a pod share an IP address and may share the volumes defined in that pod.

Kubernetes Concept: Node

NODES ARE MACHINES THAT RUN:

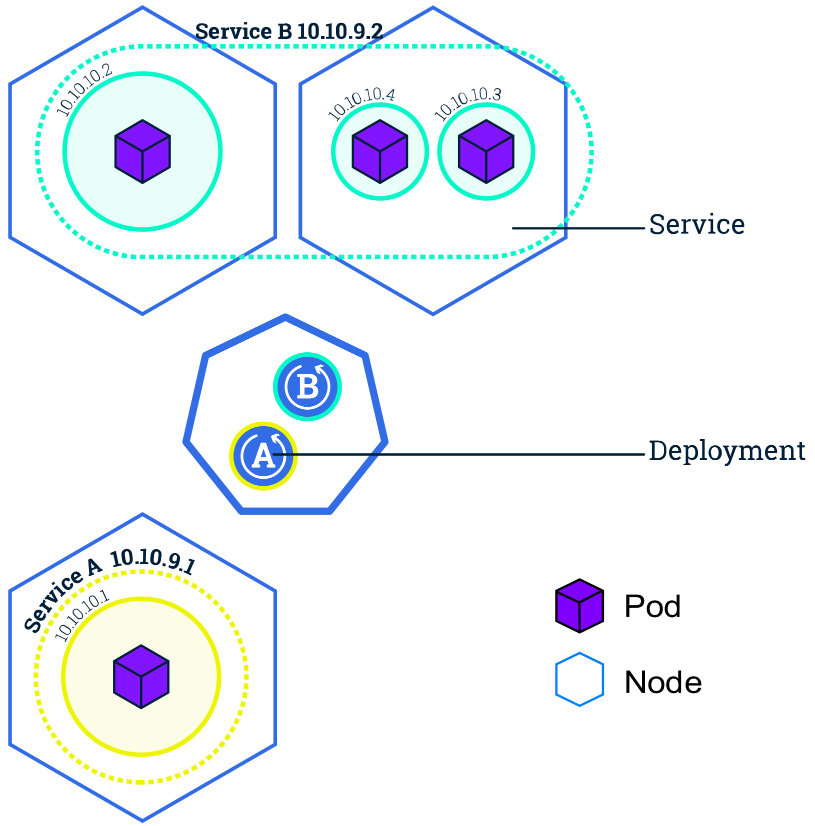
- Kubelet
- Container Runtime: Docker
- Kube-proxy



Kubernetes Concept: Service

A SERVICE IS A LOAD BALANCER

A service creates a DNS name, IP address and port that redirects to any pods matching specific labels.



Kubernetes Concept: Namespace

- A namespace is a logical grouping for Kubernetes objects:
  - Pods
  - Services
  - Accounts
  - Roles
- Namespaces may separate tenants or projects.

Kubernetes is Declarative

- Prefers “declarative” rather than “imperative” usage.
- You tell Kubernetes that you’d like five (5) copies of this application running.
- Kubernetes takes responsibility for keeping five containers staged, spread out to as many as five nodes, watching for container or node failures.
- You build YAML files describing what you want, pass these to the API server, and let it take responsibility for effecting that declaration.

```
kubectl apply -f file.yaml
```

Kubernetes Target Components (1 of 2)

- Kubernetes API Server
  - Accepts the declarative configurations and directs other components to take action.
- Kubelet
  - Runs on each node in the cluster, bridging the Kubernetes infrastructure to the container runtime (often Docker)
- Container Runtime/Docker
  - Pulls container images and instructs the kernel to start up containers

Kubernetes Target Components (2 of 2)

- ETCD Server
  - Retains the state of the cluster
- Kubernetes Dashboard
  - Web interface that permits configuration and administration
- Metrics Components
  - Provide useful data about the target

Attacking Kubernetes Clusters

- An attack on Kubernetes generally starts from the perspective of a compromised pod.
  - The threat actor may have compromised the application running in one container in the pod.
  - The threat actor may have phished/compromised a person who had access to the pod.
  - The threat actor may be a user who is looking to escalate their privileges.

Threat Actor Actions

- An attacker in a pod may:
  - Use the access provided by the pod to access other services
  - Attack other containers in their pod
  - Make requests to the API server or a Kubelet to:
    - Run commands (possibly interactively) in a different pod
    - Start a new pod with privilege and node filesystem/resource access
  - Gather secrets that Kubernetes provides to pods
  - Connect to the Kubernetes Dashboard to perform actions
  - Interact with the etcd server to change the cluster state
  - Interact with the cloud service provider using the cluster owner’s account.

RSA Conference2019

Attack Demonstrations

Attack Demo

- We'll compromise a Kubernetes cluster, starting from a vulnerable application, running in a pod on the cluster.
- This cluster is called Bust-a-Kube. You can download it with the link provided in the video.
- To recreate the attack, put Bust-a-Kube into the “Cluster Takeover 1” scenario.

Dissecting the Attack Demo

- We achieved RCE in the frontend pod and ran a Meterpreter.
- We interacted with the API server and tried to stage a pod.
- We moved laterally to a Redis pod, which had a better role.
- We staged a custom pod with a hostPath mount onto a node, compromising it.
- We staged pods to every node using a Daemon Set, compromising every one.

Demo: Multitenant Attack

- In this video demo, we'll attack a Kubernetes cluster that has a soft multitenancy setup, with a Marketing department and a Development department.
- You can recreate this demo by putting Bust-a-Kube into the "Multi-tenancy Escape 1" scenario.

Dissecting the Multitenant Attack (1 of 2)

- Gained a Meterpreter in Marketing's Wordpress container. (Flag 1)
- Moved into Marketing's MySQL container (Flag 2)
- Used the MySQL container's unfettered network access to reach a Kubelet on the master node.
- Used the Kubelet's lack of authentication to invade Development's dev-web container. (Flag 3)
- Reasoning that the dev-sync container in this same pod might be used to synchronize content, gained the pod's secrets (SSH key and account).

Dissecting the Multitenant Attack (2 of 2)

- Authenticated to the high-value Developer machine. (Flag 4)
- Returned to the cluster, used the dev-web pod's placement on the master to gain control of the AWS account. (Bonus)

RSAConference2019

Defenses

Breaking the Attack Paths!

Overarching Defense: Upgrade!

- You must upgrade your Kubernetes cluster.
- Kubernetes development is moving very quickly, with many of the features we're about to discuss only moving out of alpha or beta in the last few major releases.
- Default settings in Kubernetes (and its third-party installers) continue to strengthen substantially.
- Support periods (patching) here resembles the world of smart phones far more than the world of desktop operating systems.

Defense for Flag 3: Kubelet Authorization

- Enable the NodeRestriction admission plugin to prevent a kubelet on a node from modifying other nodes.
- The API server must include `--authorization-mode=Node`.
- Reference:

  https://kubernetes.io/docs/admin/authorization/node/

Defense #2 for Flag 3: Network Policies

- Let’s create as default deny egress network policy to prevent a container from communicating with the Kubelet or any part of our control plane.

Network Policies

- Network policies let you set firewall rules, using label selection.
- You create one or more policies.
- Each policy names pods that it refers to via a podSelector.
- Rules are for ingress and/or egress.
- Once you create a network policy for a pod, you have a default deny for traffic for that pod in that direction.

```
kind: NetworkPolicy
apiVersion:
networking.k8s.io/v1
metadata:
  name: yourpolicy
  namespace: yourns
spec:
  podSelector:
  ingress:
  egress:
```

Network Policy Example

- This policy allows traffic **IN** to pods with labels:

  ```
  app    : myapp
  role   : api
  ```

- It permits traffic only from pods with label app set to myapp.
- These labels have no inherent meaning.

```
kind: NetworkPolicy
apiVersion:
networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: myapp
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: myapp
```

Defense for Multitenant Flag 2: RBAC

- You can place restrictions on the API server via RBAC.
- Requests looks like:
  - Username (Subject)
    - Ex: [jay in group system:authenticated]
  - Verb
    - Ex: [in inguardians-ns, get pods]

- You provide the ability to do these things by creating:
  - Role/ClusterRole
  - RoleBinding

RBAC: Example

Role-pod-getter.yaml

```
kind: Role
apiVersion: …
metadata:
   name: ing-pod-getter
   namespace: inguardians-ns
rules:
- verbs: ["get"]
  apiGroups: [""]
  resources: ["pods"]
```

binding-jay-to-role.yaml

```
kind: RoleBinding
apiVersion: …
metadata:
   name: jay-pod-getter
   namespace: inguardians-ns
roleRef:
  kind: Role
  apiGroup: …
  name: ing-pod-getter
Subjects:
- kind: User
  apiGroup: …
  name: jay
```

Creating RBAC Roles Automatically

- Jordan Liggitt wrote a tool called Audit2RBAC, similar to Audit2Allow for SELinux.

  https://github.com/liggitt/audit2rbac/

- Let's see a demo video.

Defense Against Cluster Compromise: AppArmor

- In the attack, we took over the nodes by adding a pod to each one that mounted the host's filesystem as a volume.
- Let's use AppArmor to prevent the attack pods from writing to files in the host's filesystem.

- We enforce AppArmor profiles on pods via Pod Security Policies.

Pod Security Policies

- Pod Security Policies allow you to restrict the privilege with which a pod runs.
  - Volume white-listing / Usage of the node's filesystem
  - Read-only root filesystem
  - Run as a specific (non-root) user
  - Prevent privileged containers (all capabilities, all devices, …)
  - Root capability maximum set
  - SELinux or AppArmor profiles – choose from a set
  - Seccomp maximum set

Pod Security Policy: Root Capability Supersets

- All of the "magic powers" that the root user has are named and numbered, codified in a POSIX standard called "capabilities."
- Here are a few of the most common ones that containers still maintain:
  - NET_BIND_SERVICE - Bind to TCP/UDP privileged ports (<1024).
  - DAC_OVERRIDE - Bypass file read, write & execute permission checks
  - CHOWN - Make arbitrary changes to file UIDs and GIDs
  - SETUID - Make arbitrary manipulations of process UIDs
  - KILL - Bypass permission checks for sending signals

Seccomp

- Similarly to the root capabilities, you can enforce a system call whitelist on pods that are deployed in your cluster.
- This locks the set of system calls to the ones the containerized program used when uncompromised.
- This has two purposes:
  - Restrict what a compromised program can do
  - Reduce the kernel's attack surface
- Kubernetes can require that any pod running must a seccomp filter from a set that the cluster administrators vet.

Center for Internet Security Benchmark

- You can find many hardening steps for a Kubernetes cluster in the Center for Internet Security's benchmark document for Kubernetes.

  https://www.cisecurity.org/benchmark/kubernetes/

- Kube-Bench can check a cluster against this benchmark.

  https://github.com/aquasecurity/kube-bench

Tool Demo and Release

- InGuardians has several Kubernetes security tools.
- We’re releasing one today called Peirates (greek for "Pirates").
- We’ll demonstrate this attack tool now. Among other things, it can compromise a Kubernetes cluster.
- The project team includes: Faith Alderson, Jay Beale, Adam Crompton and Dave Mayer.
- Find it on InGuardians’ Github page.

DEMO

What Do I Do With This?

- Take these attacks and defenses to your work.
- If you're permitted, try the attacks. If they work, there's something for you to do.
- Take the defenses to:
  - Cluster Design and Maintenance
  - Application Design and Rollout