Microservices Security, Container Runtime Security, MITRE ATT&CK® for Kubernetes (K8S) and Service Mesh for Security (Demo Included!)

Nathan Aw
https://www.linkedin.com/in/awnathan

15 July 2020

This Talk

- Background - Context - Problem Statement
- Microservices 101 & Primer
- Recap - API Security
- Microservices Security
  - Kubernetes (K8S) Security
  - MITRE ATT&CK® for K8S
  - Container Runtime Security
- How to Secure Your K8S - The Cloud Native 4Cs
- Service Mesh for Microservice Security

Opinions/views expressed in the talk are solely my own and do not express the views or opinions of my employer.

Background - Context - Problem

In the last meetup, we focused on APIs Security. APIs are the front door to Microservices. Today we focus on Microservices Security.

The Microservices Architecture/ Paradigm has special security considerations due to:

(1) tremendous increase in the number of components

(2) complex network environments comprised of various interaction styles among these components.

The attributes...

Decoupled Components

Increased Complexity

Polyglot Programming/ Architecture

And the Security Implications...

Many components to track

Many communication styles (e.g., REST), protocols (e.g., HTTP) and data formats (e.g., JSON)

https://owasp.org/www-chapter-singapore/assets/presos/Securing_your_APIs_-_OWASP_API_Top_10_2019,_Real-life_Case.pdf

Who I am. Hello.

- Currently an **AppDevSec** Digital Solutions Architect and a Full-Stack Developer in the Financial Services Industry (FSI).
  - First a Full-Stack Cloud-Native Developer, then a Solutions Architect
  - Previously worked in a local bank as a Full-Stack Blockchain Engineer
  - *Have Designed, Built, Deployed and Operated* ***> 58*** *Unique Polyglot Based Production Grade Microservices (Micro Frontends, Backend for Frontends, Backends) over last 3 years*
- **Specialties** around API, Microservices that enables a Seamless & Frictionless Customer Journey Experience (CJX)
  - On "Hybrid-Multi" Cloud Native Platforms
  - On API, Microservices Security, Container Runtime Security and MITRE ATT&CK® for Kubernetes(K8S)
- Technology Stack: Golang, React, Kafka, Spring Boot, NodeJS, Apigee, Kong, Zuul, GraphQL, Azure Kubernetes Service (AKS), Elastic Kubernetes Service (EKS), Openshift, Service Mesh (Istio, Linkerd, Envoy), Cloud Foundry, GraphQL and many more…
- Designing, building and operating Scalable, Secure and Robust APIs and Microservices is my passion!
- https://www.linkedin.com/in/awnathan

What are Microservices? And what are its goals?

1

- Functional system decomposed/deconstructed into manageable and independently deployable components

- Functional system decomposition implies vertical slicing (versus horizontal slicing through layer)

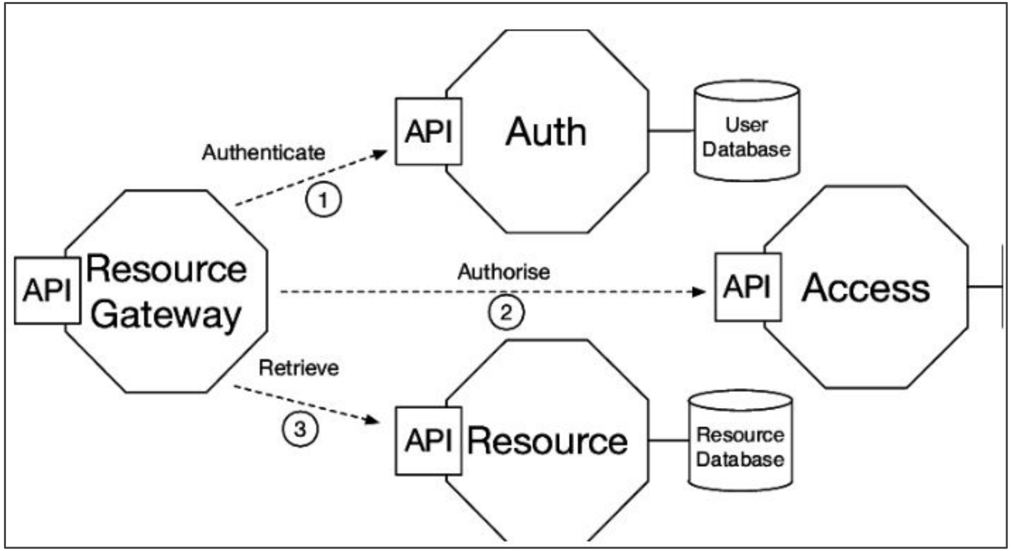
- Independent deployability implies **no** shared state and inter-process communication via HTTP RESTful interface

2

Independent deployability is the objective.

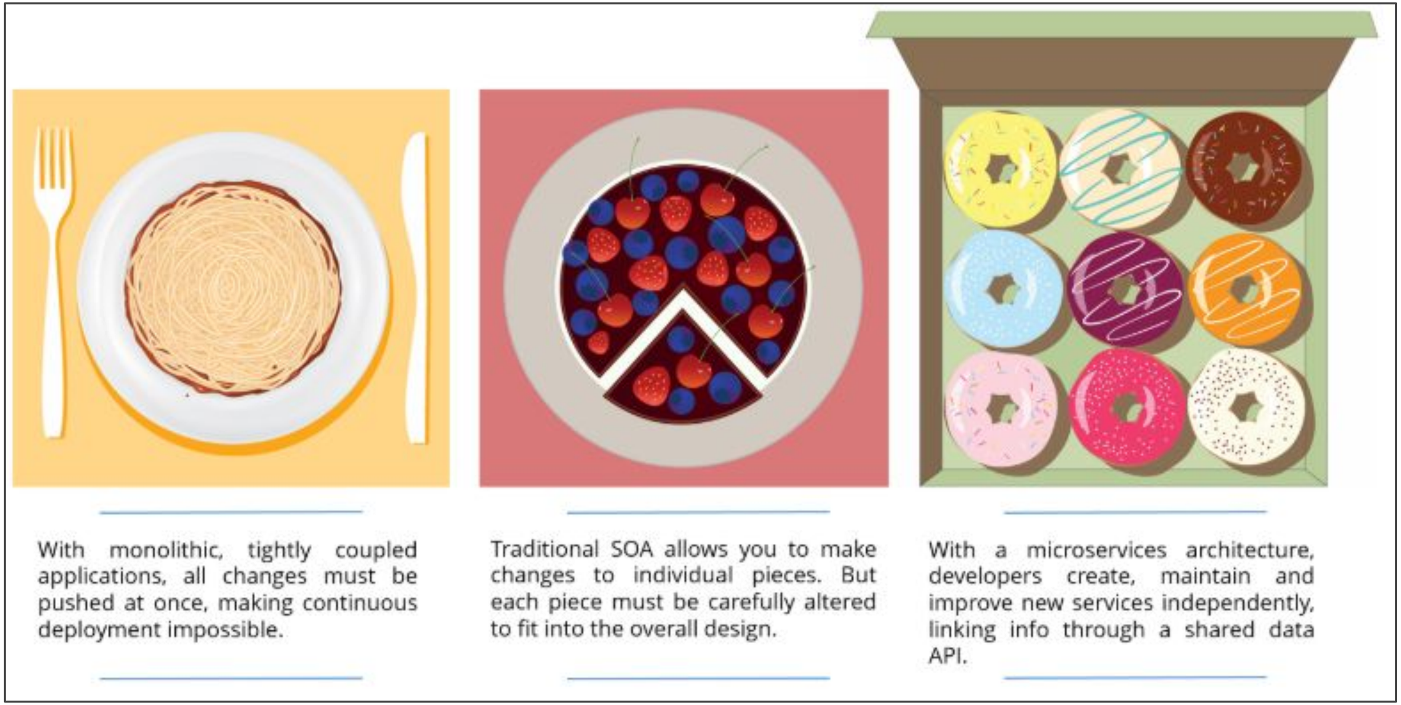
3

Business Agility as the outcome.



SOURCE: https://www.researchgate.net/figure/Example-of-Microservice-Architecture_fig1_305881421

An Illustration of Microservices Architecture (1/2)



With monolithic, tightly coupled applications, all changes must be pushed at once, making continuous deployment impossible.

Traditional SOA allows you to make changes to individual pieces. But each piece must be carefully altered to fit into the overall design.

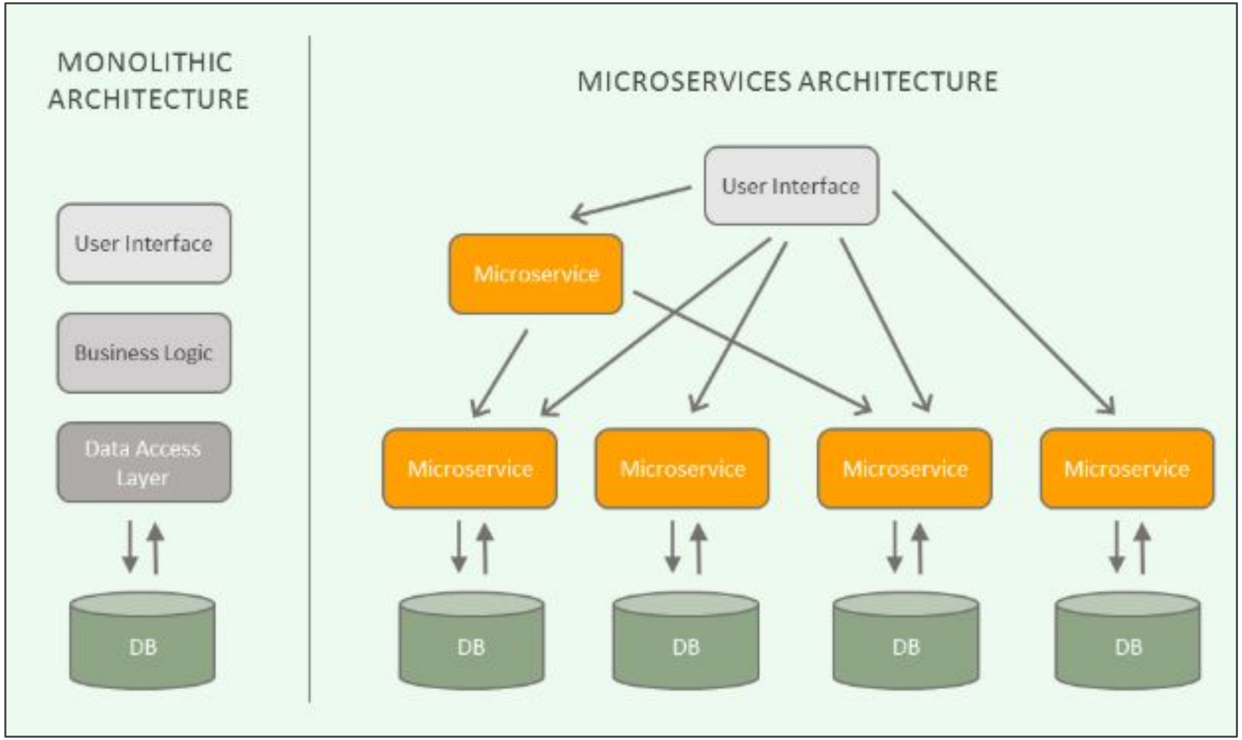
With a microservices architecture, developers create, maintain and improve new services independently, linking info through a shared data API.

“Enables developers to use **different programming language**, depending on what they believe is the best one for the specific business function the microservice is built around.”

Independent deployability is the objective.

SOURCE: https://dzone.com/articles/what-are-microservices-actually

Sample Microservices Architecture (2/2)



“Allow developers to build their applications from various independent components which can easily be changed, removed or upgraded without affecting the whole application – as is not the case with monoliths.”

Independent deployability is the objective.

SOURCE: https://dzone.com/articles/what-are-microservices-actually

Microservices - Not a silver bullet; Multiple Tradeoffs including *“Perrow-ian”* Complexity*.

“Microservices are a great pattern when they map services to disparate teams that deliver them, **or** when the value of independent rollout and the value of independent scale are greater than the cost of orchestration.” - Istio

“Value of independent rollout + value of independent scale.”

>

Cost of

Orchestration.

=

Microservices can be considered.

“The ‘Interactive Complexity’ associated with a fundamentally distributed environment that might result in cascading failure must be the foremost consideration.” - Nathan Aw

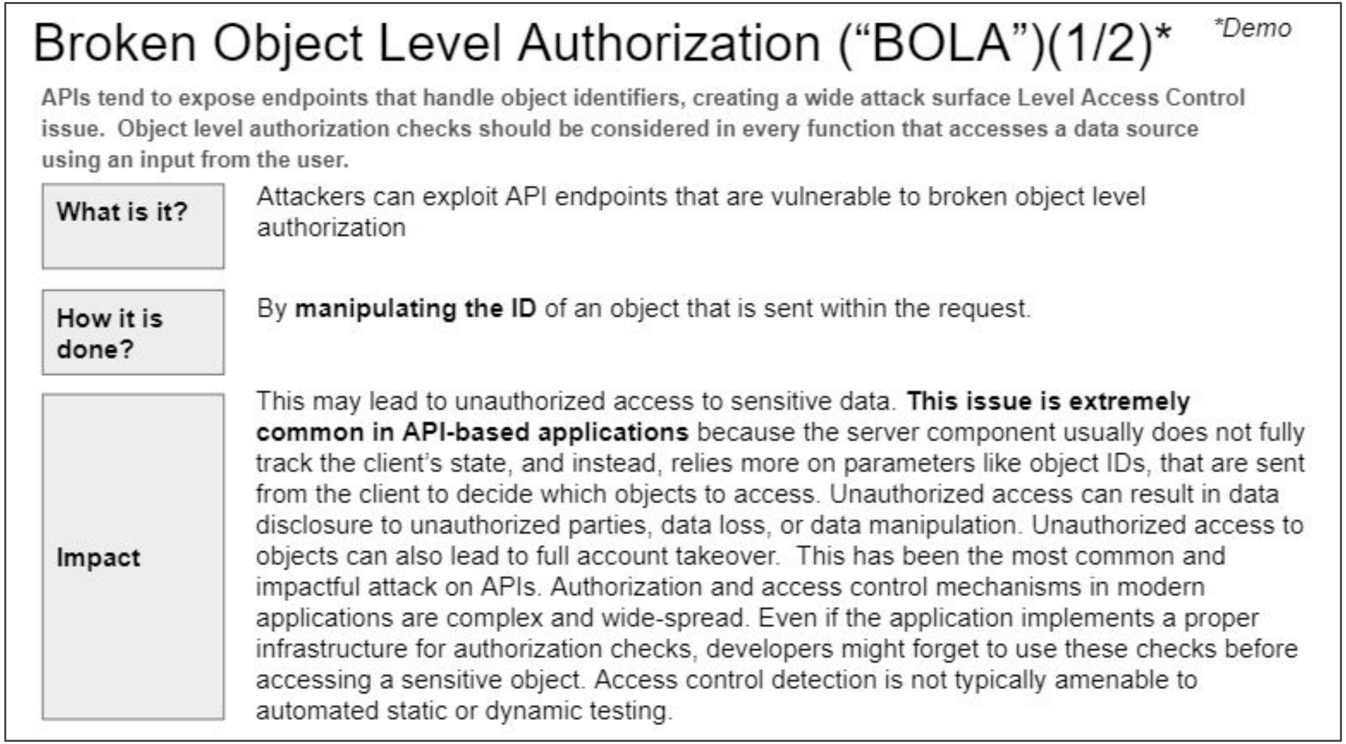
SOURCE: https://en.wikipedia.org/wiki/Conway%27s_law; *https://www.oreilly.com/radar/cloud-adoption-in-2020/; https://istio.io/latest/blog/2020/istiod/ ; https://en.wikipedia.org/wiki/System_accident

Recap - Previous OWASP Meetup on API Security

“Independent deployability” also implies…

(1) **no** shared state - stateless

(2) inter-process communication via RESTful interface (HTTP)



Broken Object Level Authorization (“BOLA”)(1/2)*

*Demo

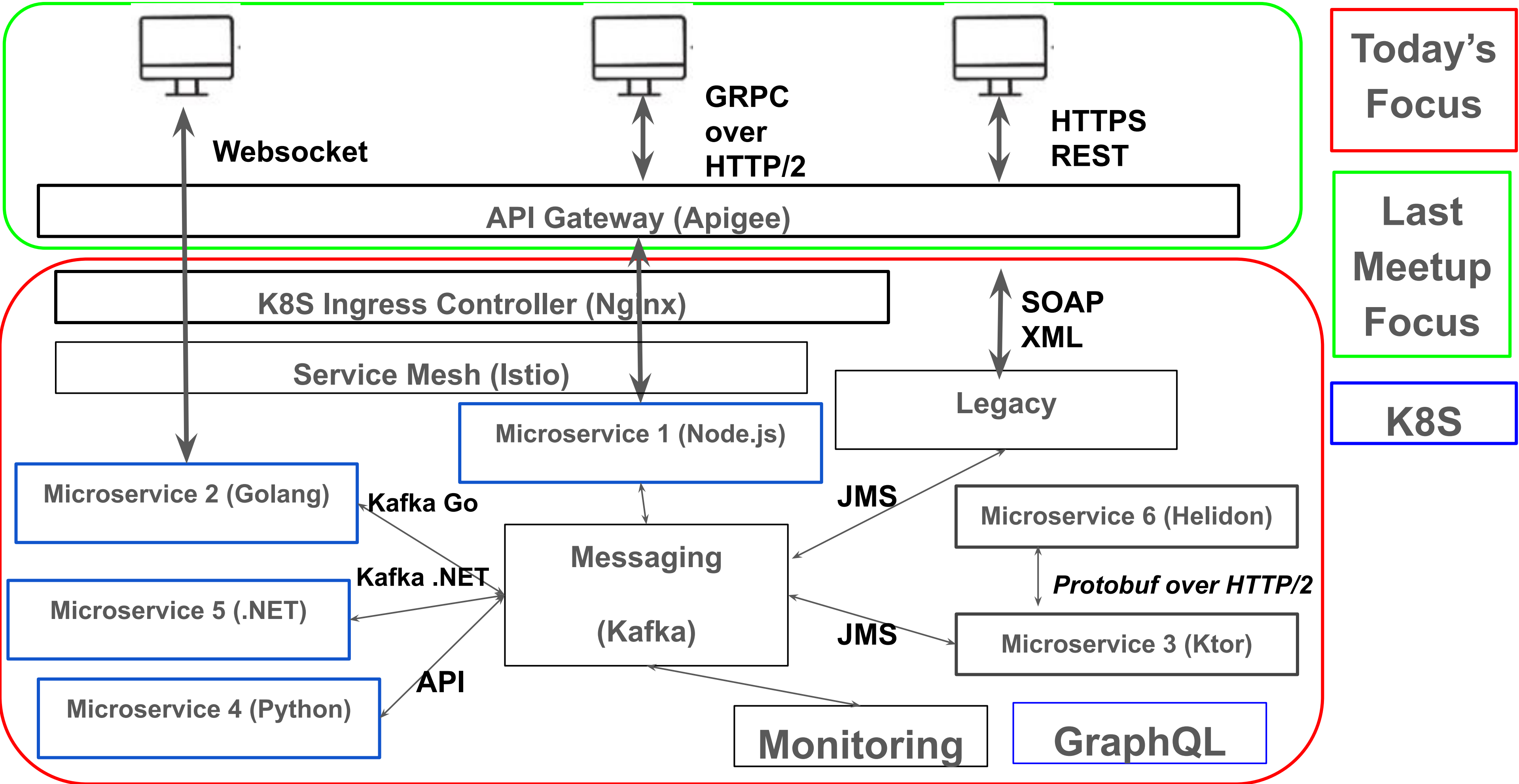
APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.

What is it?	Attackers can exploit API endpoints that are vulnerable to broken object level authorization
How it is done?	By **manipulating the ID** of an object that is sent within the request.
Impact	This may lead to unauthorized access to sensitive data. **This issue is extremely common in API-based applications** because the server component usually does not fully track the client’s state, and instead, relies more on parameters like object IDs, that are sent from the client to decide which objects to access. Unauthorized access can result in data disclosure to unauthorized parties, data loss, or data manipulation. Unauthorized access to objects can also lead to full account takeover. This has been the most common and impactful attack on APIs. Authorization and access control mechanisms in modern applications are complex and wide-spread. Even if the application implements a proper infrastructure for authorization checks, developers might forget to use these checks before accessing a sensitive object. Access control detection is not typically amenable to automated static or dynamic testing.

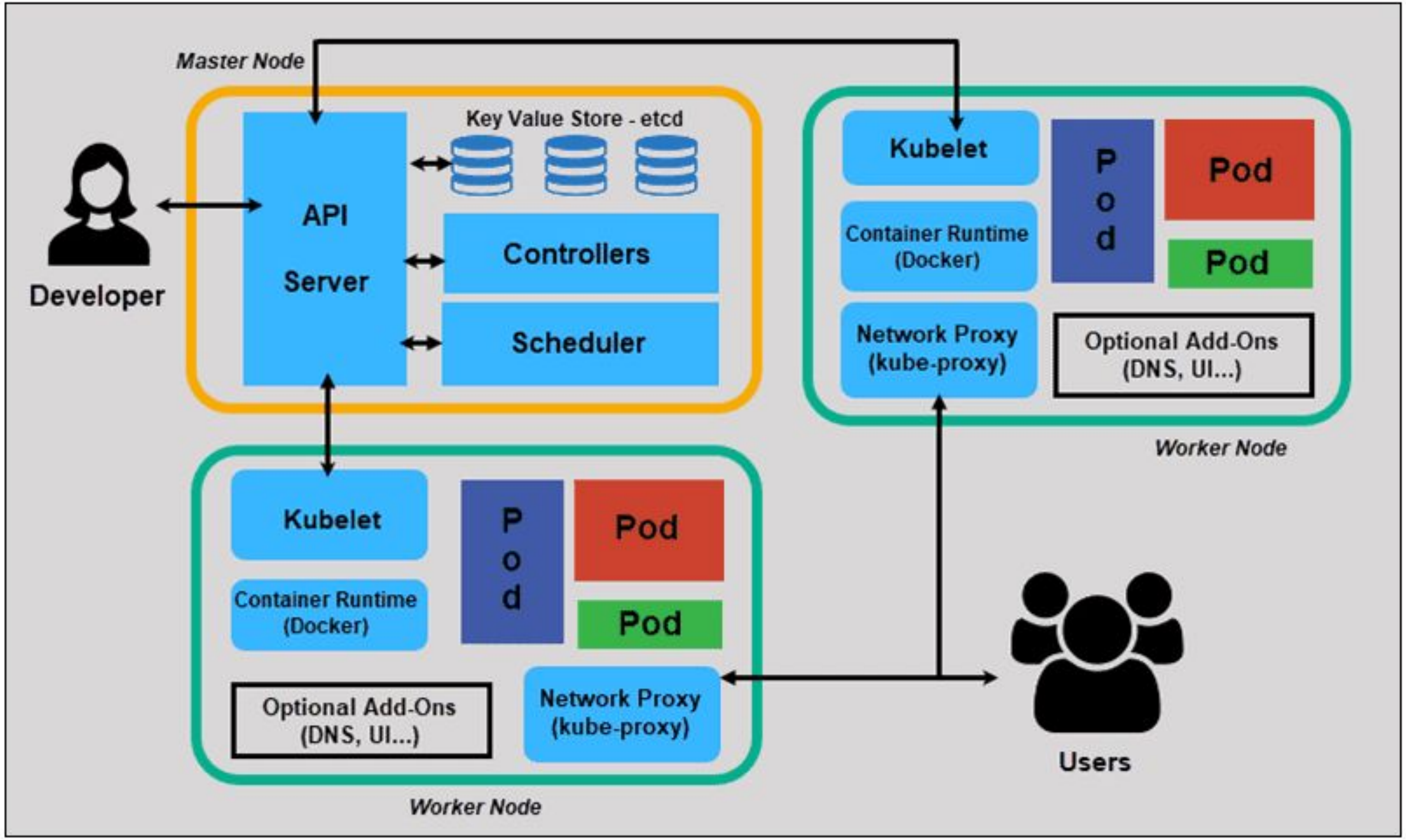
“The interplay between Microservices Security and APIs Security needs to be very carefully considered and examined.” - Nathan Aw

SOURCE: https://owasp.org/www-chapter-singapore/assets/presos/Securing_your_APIs_-_OWASP_API_Top_10_2019,_Real-life_Case.pdf

Sample (Actual) Polyglot Microservices Architecture - *Highly* Simplified



Kubernetes Architecture



SOURCE: https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams

Microservices Security (1/4) - Mere Snapshot of the Sprawling Landscape!

Microservices

Landscape

(A small snapshot)

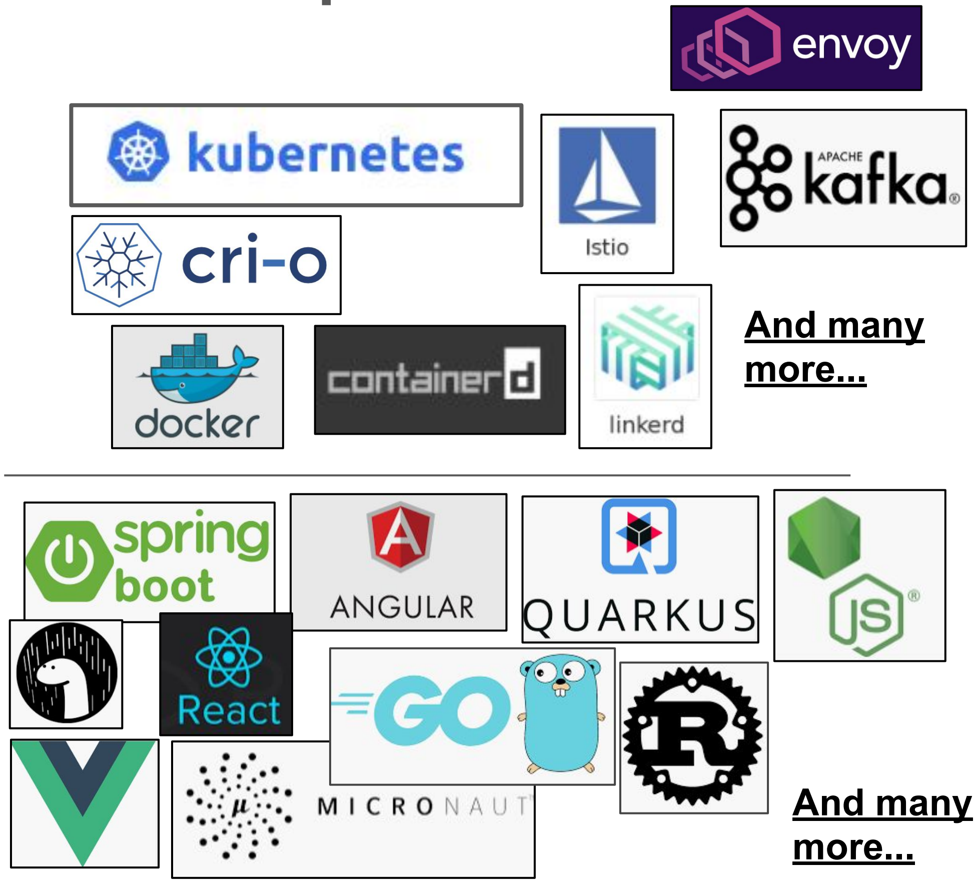
Infrastructure

(Container Runtime, Orchestration, Messaging, Mesh, etc)

Programming

Frameworks

(Polyglot)

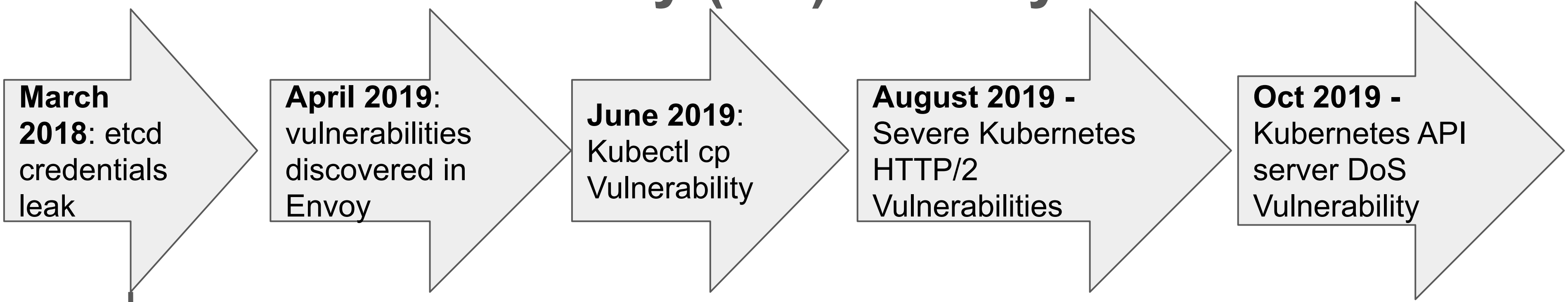


And many more...

And many more...

Node.js, Deno, Golang, Rust, Quarkus, Micronaut and Vue.js are my personal favourites - ping me up to ask why!

Microservices Security (2/4) - Today's Situation



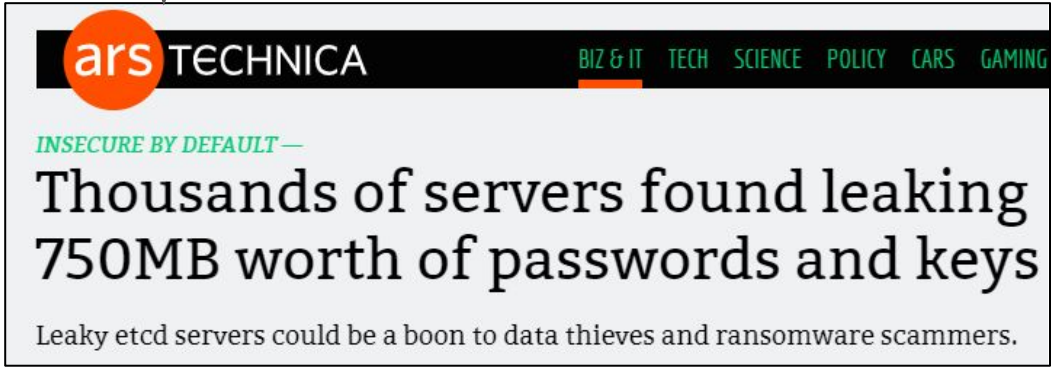
March 2018: etcd credentials leak

April 2019: vulnerabilities discovered in Envoy

June 2019: Kubectl cp Vulnerability

August 2019 - Severe Kubernetes HTTP/2 Vulnerabilities

Oct 2019 - Kubernetes API server DoS Vulnerability

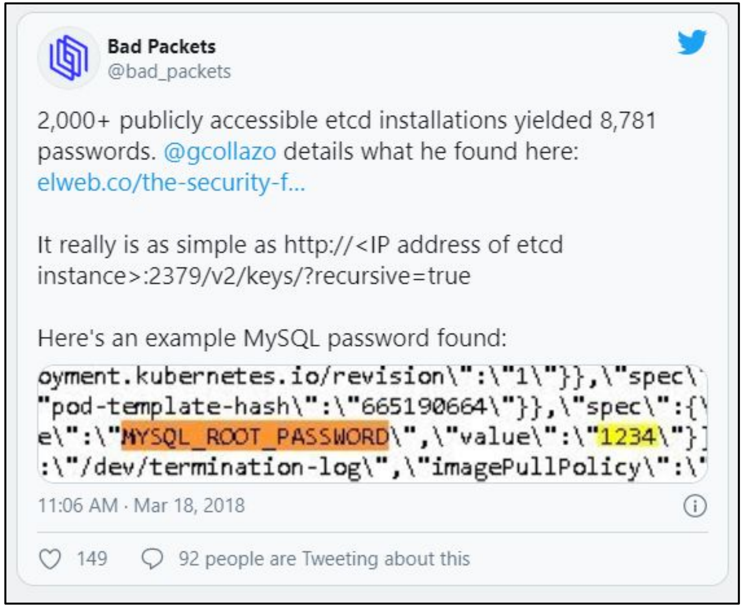


INSECURE BY DEFAULT —

Thousands of servers found leaking 750MB worth of passwords and keys

Leaky etcd servers could be a boon to data thieves and ransomware scammers.

SOURCE: Sysdig "Securing Kubernetes in Production"; https://arstechnica.com/information-technology/2018/03/thousands-of-servers-found-leaking-750-mb-worth-of-passwords-and-keys/



Bad Packets
@bad_packets

2,000+ publicly accessible etcd installations yielded 8,781 passwords. @gcollazo details what he found here: elweb.co/the-security-f...

It really is as simple as http://<IP address of etcd instance>:2379/v2/keys/?recursive=true

Here's an example MySQL password found:

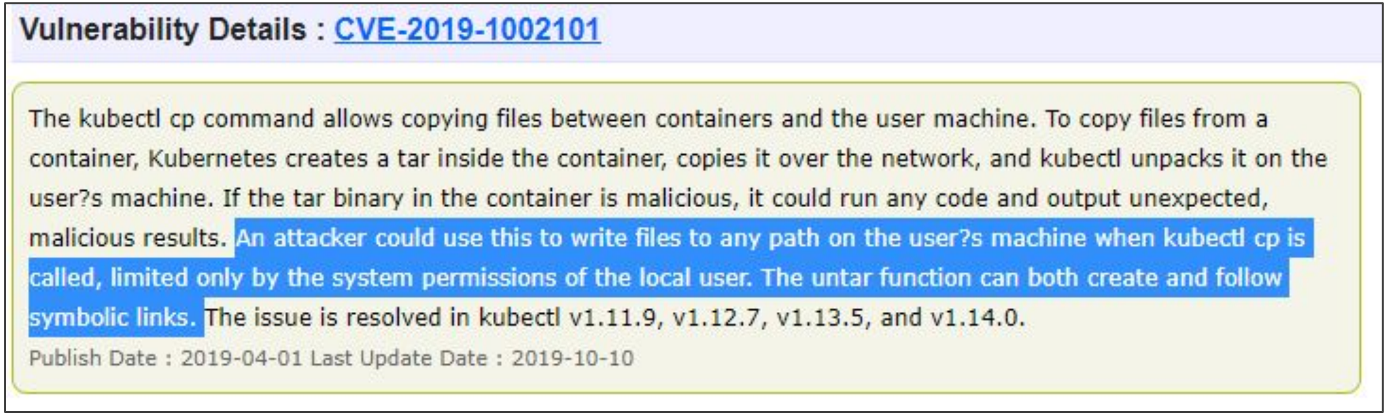
```
oyment.kubernetes.io/revision\":\"1\"}},\"spec\'
"pod-template-hash\":\"665190664\"}},\"spec\":{\
e\":\"MYSQL_ROOT_PASSWORD\",\"value\":\"1234\"}]
:\"/dev/termination-log\",\"imagePullPolicy\":\'
```

11:06 AM · Mar 18, 2018

♡ 149 92 people are Tweeting about this

Microservices Security (3/4) - Today’s Situation

June 2019: Kubectl cp Vulnerability

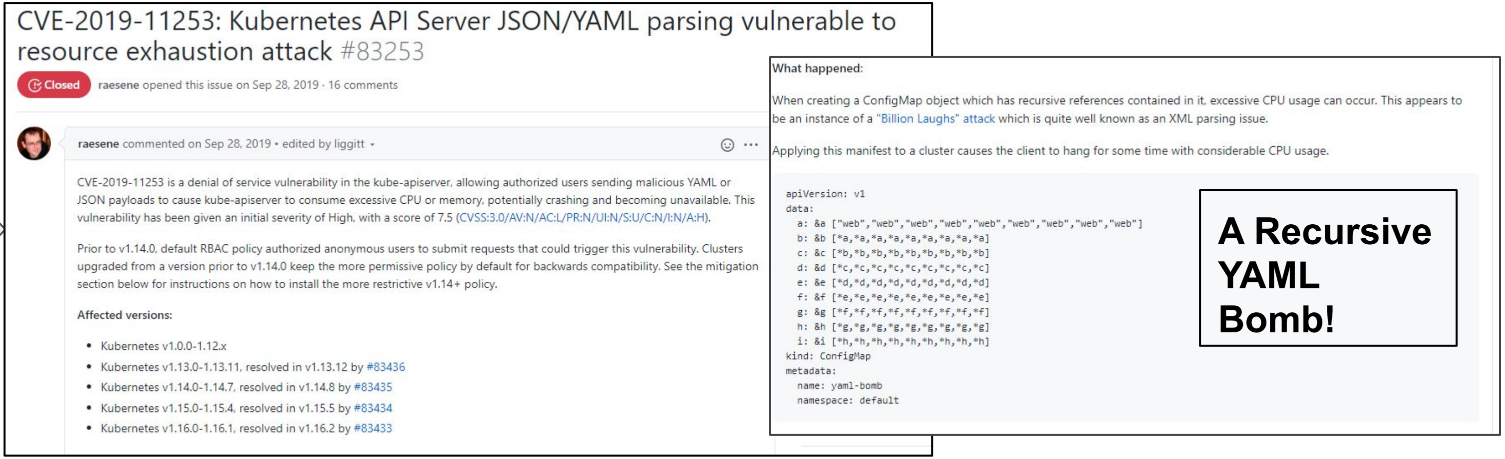


Vulnerability Details : CVE-2019-1002101

The kubectl cp command allows copying files between containers and the user machine. To copy files from a container, Kubernetes creates a tar inside the container, copies it over the network, and kubectl unpacks it on the user?s machine. If the tar binary in the container is malicious, it could run any code and output unexpected, malicious results. An attacker could use this to write files to any path on the user?s machine when kubectl cp is called, limited only by the system permissions of the local user. The untar function can both create and follow symbolic links. The issue is resolved in kubectl v1.11.9, v1.12.7, v1.13.5, and v1.14.0.

Publish Date : 2019-04-01 Last Update Date : 2019-10-10

Oct 2019 - Kubernetes API server DoS Vulnerability



CVE-2019-11253: Kubernetes API Server JSON/YAML parsing vulnerable to resource exhaustion attack #83253

raesene opened this issue on Sep 28, 2019 · 16 comments

raesene commented on Sep 28, 2019 • edited by liggitt

CVE-2019-11253 is a denial of service vulnerability in the kube-apiserver, allowing authorized users sending malicious YAML or JSON payloads to cause kube-apiserver to consume excessive CPU or memory, potentially crashing and becoming unavailable. This vulnerability has been given an initial severity of High, with a score of 7.5 (CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H).

Prior to v1.14.0, default RBAC policy authorized anonymous users to submit requests that could trigger this vulnerability. Clusters upgraded from a version prior to v1.14.0 keep the more permissive policy by default for backwards compatibility. See the mitigation section below for instructions on how to install the more restrictive v1.14+ policy.

Affected versions:

- Kubernetes v1.0.0-1.12.x
- Kubernetes v1.13.0-1.13.11, resolved in v1.13.12 by #83436
- Kubernetes v1.14.0-1.14.7, resolved in v1.14.8 by #83435
- Kubernetes v1.15.0-1.15.4, resolved in v1.15.5 by #83434
- Kubernetes v1.16.0-1.16.1, resolved in v1.16.2 by #83433

What happened:

When creating a ConfigMap object which has recursive references contained in it, excessive CPU usage can occur. This appears to be an instance of a "Billion Laughs" attack which is quite well known as an XML parsing issue.

Applying this manifest to a cluster causes the client to hang for some time with considerable CPU usage.

```
apiVersion: v1
data:
  a: &a ["web","web","web","web","web","web","web","web","web"]
  b: &b [*a,*a,*a,*a,*a,*a,*a,*a,*a]
  c: &c [*b,*b,*b,*b,*b,*b,*b,*b,*b]
  d: &d [*c,*c,*c,*c,*c,*c,*c,*c,*c]
  e: &e [*d,*d,*d,*d,*d,*d,*d,*d,*d]
  f: &f [*e,*e,*e,*e,*e,*e,*e,*e,*e]
  g: &g [*f,*f,*f,*f,*f,*f,*f,*f,*f]
  h: &h [*g,*g,*g,*g,*g,*g,*g,*g,*g]
  i: &i [*h,*h,*h,*h,*h,*h,*h,*h,*h]
kind: ConfigMap
metadata:
  name: yaml-bomb
  namespace: default
```

A Recursive YAML Bomb!

SOURCE: https://www.cvedetails.com/cve/CVE-2019-1002101/; https://github.com/kubernetes/kubernetes/issues/83253

Microservices Security (4/4) - Today's Situation

Vulnerabilities or Misconfigurations

52% container images fail scans with high severity* that leaves applications exposed to attacks*

Best Practices not in place and/or adhered to.

On average, 21 containers per node are running as root, opening the door for container breakouts*

Lack of Monitoring - Undetected Container Breaches

5 min container lifespan requires purpose-built tools for audit and incident response*

SOURCE: Sysdig 2019 Container Usage Report

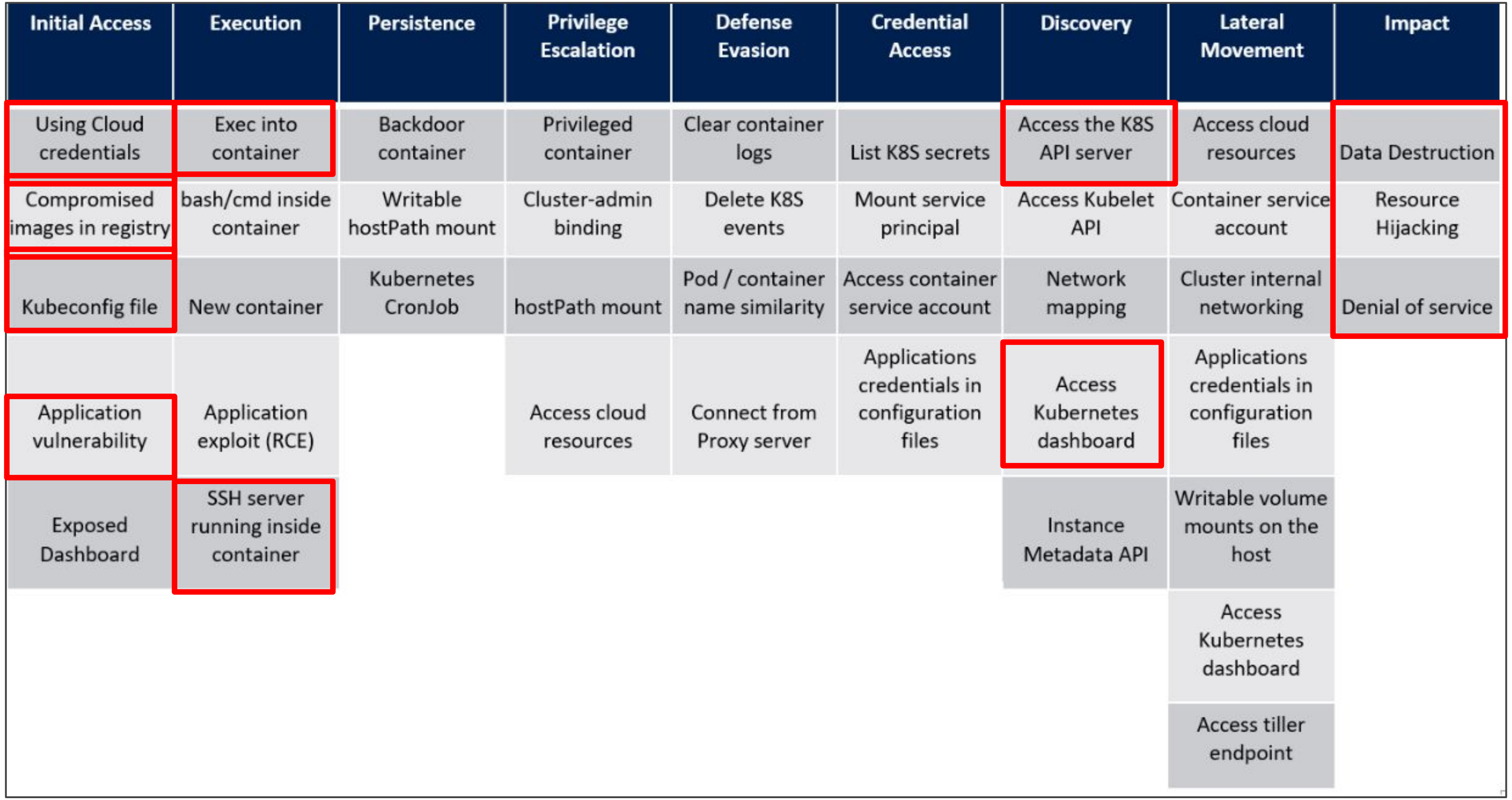
MITRE ATT&CK® Framework for Kubernetes

ATT&CK - Adversarial Tactics, Techniques, and Common Knowledge

For the uninitiated, Kubernetes(K8S) is an open source container scheduling and orchestration system.

MITRE ATT&CK® Framework for Kubernetes

ATT&CK - Adversarial Tactics, Techniques, and Common Knowledge



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

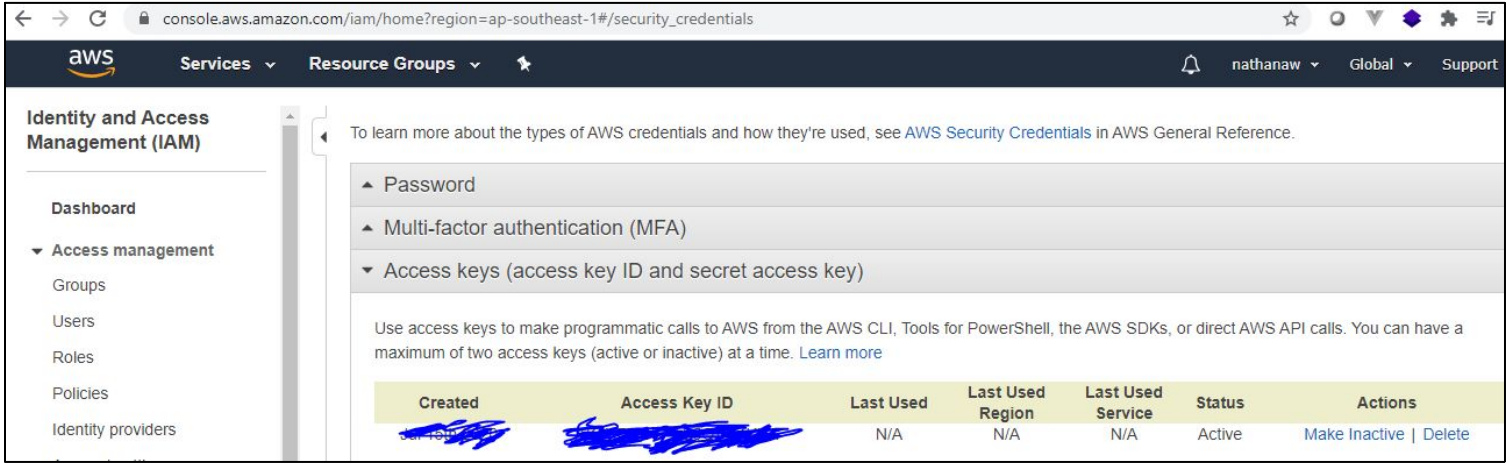
Our Focus Today

SOURCE: https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/

MITRE ATT&CK® Framework for Kubernetes

ATT&CK - Adversarial Tactics, Techniques, and Common Knowledge

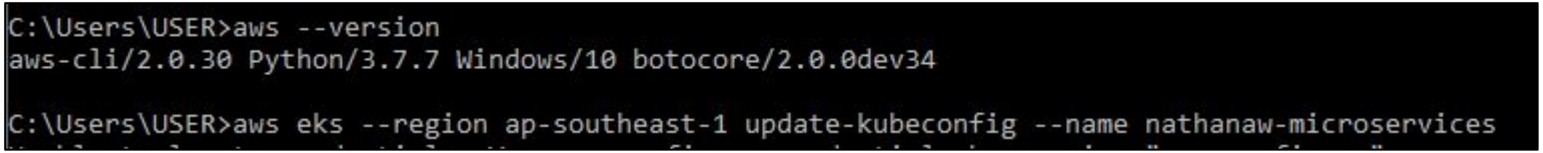
Using Cloud Credentials


console.aws.amazon.com/iam/home?region=ap-southeast-1#/security_credentials
aws
Services
Resource Groups
nathanaw
Global
Support
Identity and Access Management (IAM)
Dashboard
Access management
Groups
Users
Roles
Policies
Identity providers
Password
Multi-factor authentication (MFA)
Access keys (access key ID and secret access key)

To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.

Use access keys to make programmatic calls to AWS from the AWS CLI, Tools for PowerShell, the AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more

Created	Access Key ID	Last Used	Last Used Region	Last Used Service	Status	Actions
[redacted]	[redacted]	N/A	N/A	N/A	Active	Make Inactive | Delete



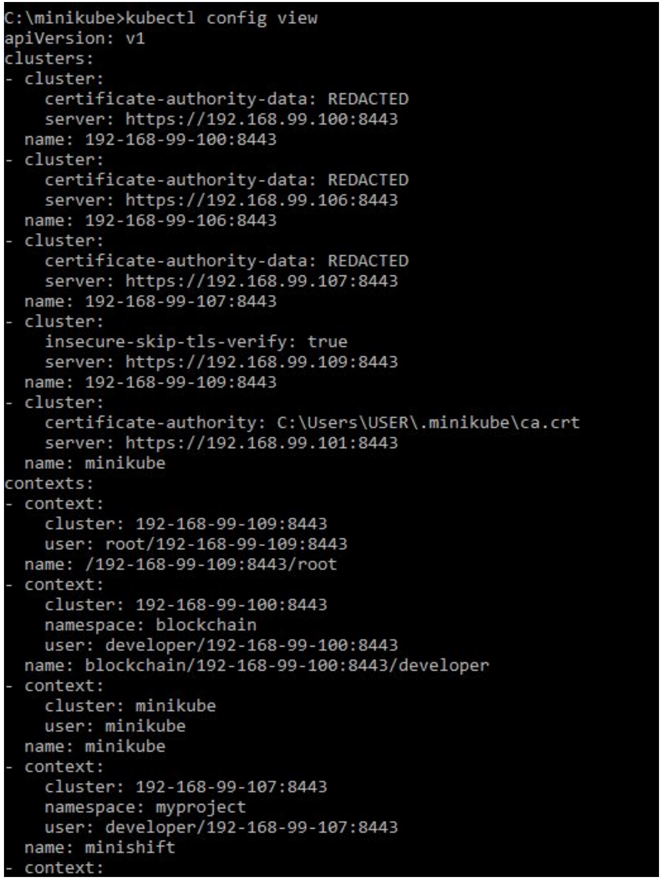
```
C:\Users\USER>aws --version
aws-cli/2.0.30 Python/3.7.7 Windows/10 botocore/2.0.0dev34

C:\Users\USER>aws eks --region ap-southeast-1 update-kubeconfig --name nathanaw-microservices
```

If your cloud credentials (e.g., AWS Root User or IAM User) are compromised, your whole Kubernetes cluster is at risk!

MITRE ATT&CK® Framework for Kubernetes

ATT&CK - Adversarial Tactics, Techniques, and Common Knowledge

Kubeconfig File



```
C:\minikube>kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://192.168.99.100:8443
  name: 192-168-99-100:8443
- cluster:
    certificate-authority-data: REDACTED
    server: https://192.168.99.106:8443
  name: 192-168-99-106:8443
- cluster:
    certificate-authority-data: REDACTED
    server: https://192.168.99.107:8443
  name: 192-168-99-107:8443
- cluster:
    insecure-skip-tls-verify: true
    server: https://192.168.99.109:8443
  name: 192-168-99-109:8443
- cluster:
    certificate-authority: C:\Users\USER\.minikube\ca.crt
    server: https://192.168.99.101:8443
  name: minikube
contexts:
- context:
    cluster: 192-168-99-109:8443
    user: root/192-168-99-109:8443
  name: /192-168-99-109:8443/root
- context:
    cluster: 192-168-99-100:8443
    namespace: blockchain
    user: developer/192-168-99-100:8443
  name: blockchain/192-168-99-100:8443/developer
- context:
    cluster: minikube
    user: minikube
  name: minikube
- context:
    cluster: 192-168-99-107:8443
    namespace: myproject
    user: developer/192-168-99-107:8443
  name: minishift
- context:
```

A kubeconfig file is a file used to configure access to Kubernetes when used in conjunction with the kubectl command line tool (or other clients).

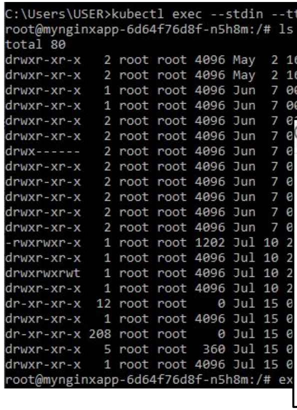
MITRE ATT&CK® Framework for Kubernetes

ATT&CK - Adversarial Tactics, Techniques, and Common Knowledge

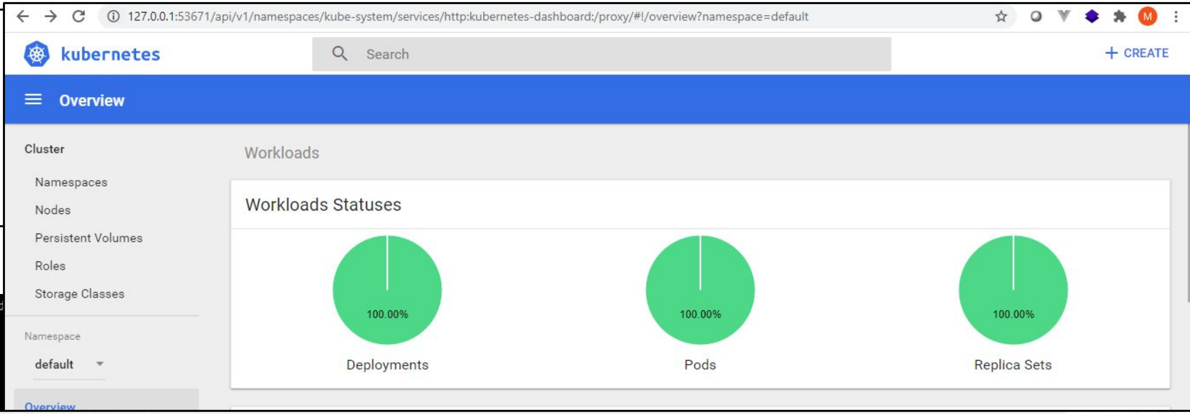
Execution into Container

Attackers who have permissions, can run malicious commands in containers in the cluster using exec command (“kubectl exec”). In this method, attackers can use legitimate images, such as an OS image (e.g., Ubuntu) as a backdoor container, and run their malicious code remotely by using “kubectl exec”.

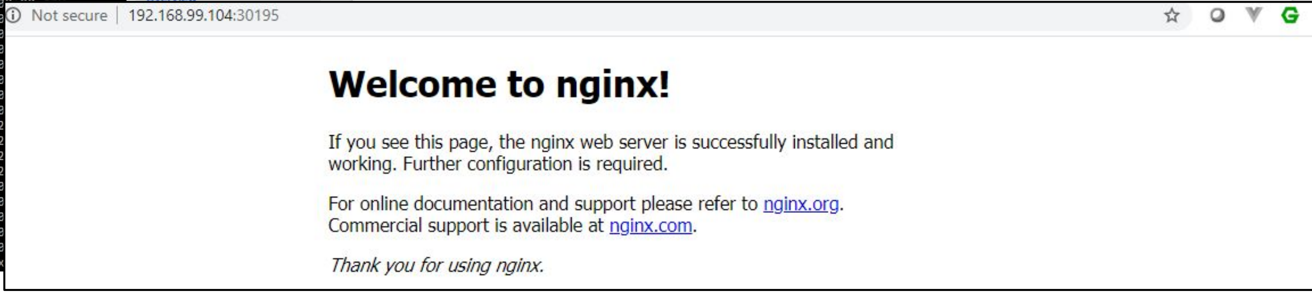
Demo (Deploy Golang + Nginx) on K8S



```
C:\Users\USER>kubectl exec --stdin --tty mynginxapp-6d
root@mynginxapp-6d64f76d8f-n5h8m:/# ls -ltr
total 80
drwxr-xr-x   2 root root 4096 May  2 16:39 home
drwxr-xr-x   2 root root 4096 May  2 16:39 boot
drwxr-xr-x   1 root root 4096 Jun  7 00:00 var
drwxr-xr-x   1 root root 4096 Jun  7 00:00 usr
drwxr-xr-x   2 root root 4096 Jun  7 0
drwxr-xr-x   2 root root 4096 Jun  7 0
drwx------   2 root root 4096 Jun  7 0
drwxr-xr-x   2 root root 4096 Jun  7 0
drwxr-xr-x   2 root root 4096 Jun  7 0
drwxr-xr-x   2 root root 4096 Jun  7 0
drwxr-xr-x   2 root root 4096 Jun  7 0
drwxr-xr-x   2 root root 4096 Jun  7 0
-rwxrwxr-x   1 root root 1202 Jul 10 2
drwxr-xr-x   1 root root 4096 Jul 10 2
drwxrwxrwt   1 root root 4096 Jul 10 2
drwxr-xr-x   1 root root 4096 Jul 10 2
dr-xr-xr-x  12 root root    0 Jul 15 0
drwxr-xr-x   1 root root 4096 Jul 15 0
dr-xr-xr-x 208 root root    0 Jul 15 0
drwxr-xr-x   5 root root  360 Jul 15 0
drwxr-xr-x   1 root root 4096 Jul 15 0
root@mynginxapp-6d64f76d8f-n5h8m:/# ex
```



127.0.0.1:53671/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/#!/overview?namespace=default



Not secure | 192.168.99.104:30195

Welcome to nginx!

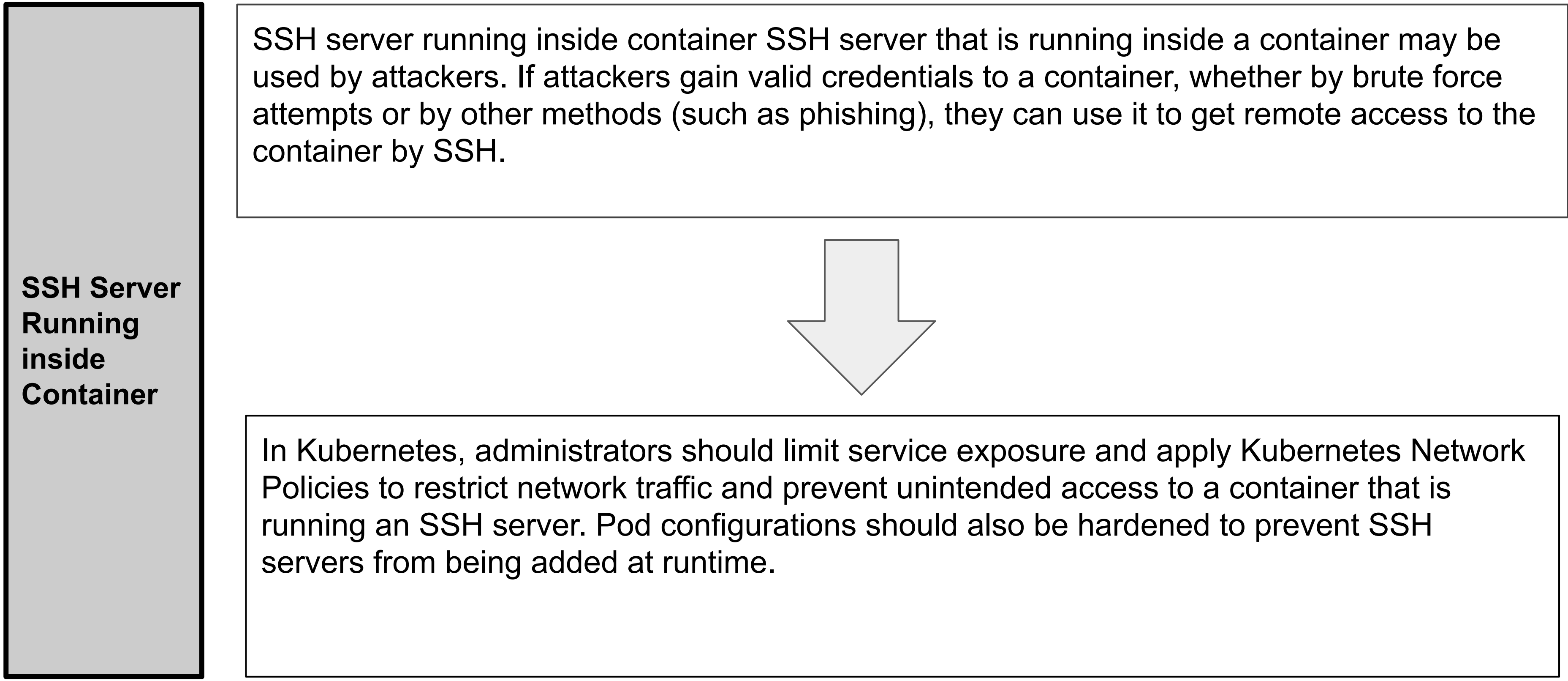
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

MITRE ATT&CK® Framework for Kubernetes

ATT&CK - Adversarial Tactics, Techniques, and Common Knowledge



SSH Server Running inside Container

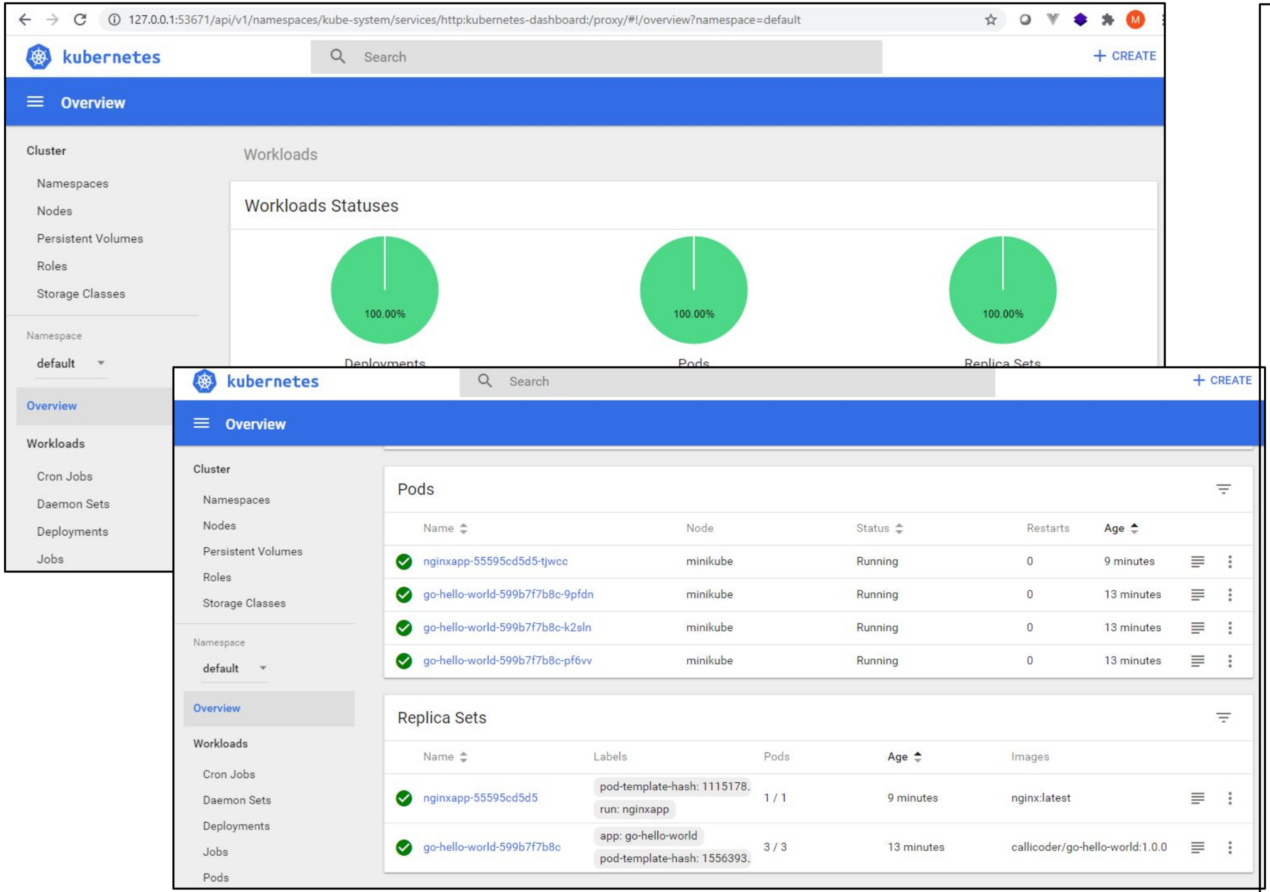
SSH server running inside container SSH server that is running inside a container may be used by attackers. If attackers gain valid credentials to a container, whether by brute force attempts or by other methods (such as phishing), they can use it to get remote access to the container by SSH.

In Kubernetes, administrators should limit service exposure and apply Kubernetes Network Policies to restrict network traffic and prevent unintended access to a container that is running an SSH server. Pod configurations should also be hardened to prevent SSH servers from being added at runtime.

SOURCE: https://www.stackrox.com/post/2020/07/protecting-against-kubernetes-threats-chapter-2-execution/

MITRE ATT&CK® Framework for Kubernetes (2/3)

Access Kubernetes Dashboard



Pods

Name	Node	Status	Restarts	Age
✓ nginxapp-55595cd5d5-tjwcc	minikube	Running	0	9 minutes
✓ go-hello-world-599b7f7b8c-9pfdn	minikube	Running	0	13 minutes
✓ go-hello-world-599b7f7b8c-k2sln	minikube	Running	0	13 minutes
✓ go-hello-world-599b7f7b8c-pf6vv	minikube	Running	0	13 minutes

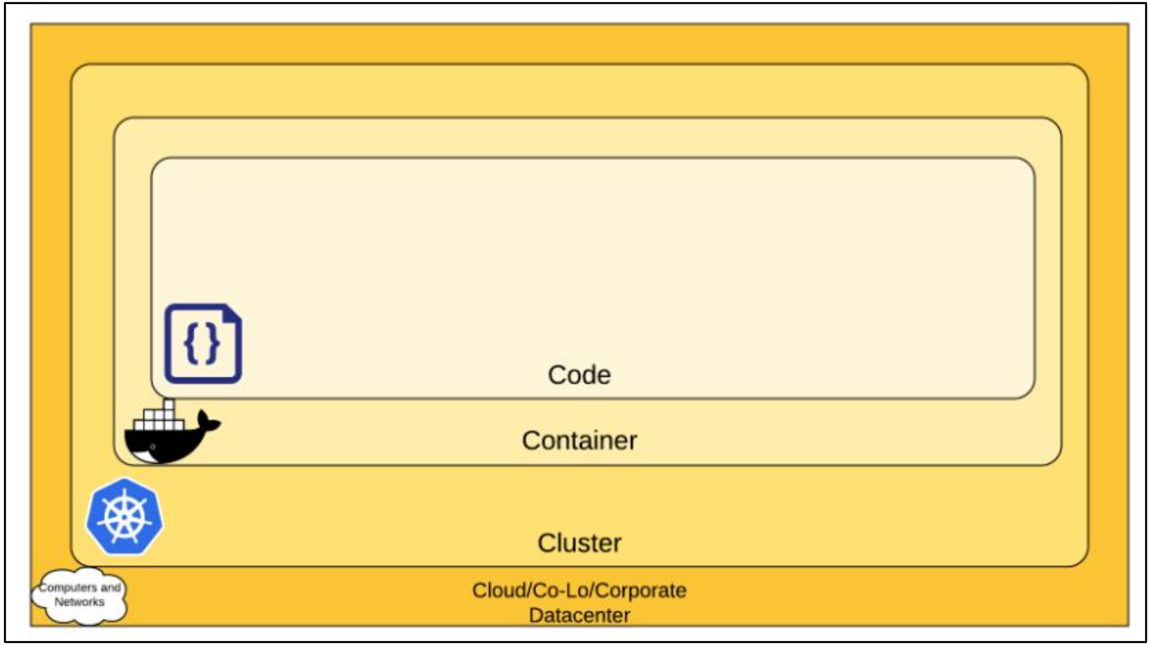
Replica Sets

Name	Labels	Pods	Age	Images
✓ nginxapp-55595cd5d5	pod-template-hash: 1115178. run: nginxapp	1 / 1	9 minutes	nginx:latest
✓ go-hello-world-599b7f7b8c	app: go-hello-world pod-template-hash: 1556393.	3 / 3	13 minutes	callicoder/go-hello-world:1.0.0

The Kubernetes dashboard is a web-based UI that is used for monitoring and managing the Kubernetes cluster. The dashboard allows users to perform actions in the cluster using its service account (kubernetes-dashboard) with the permissions that are determined by the binding or cluster-binding for this service account. Attackers who gain access to a container in the cluster, can use its network access to the dashboard pod. Consequently, attackers may retrieve information about the various resources in the cluster using the dashboard's identity.

SOURCE: http://127.0.0.1:62823/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/

How to Secure Your K8S - The Cloud Native 4Cs



1. The 4C's of Cloud Native security. You can think about security in layers.

2. The 4C's of Cloud Native security are Cloud, Clusters, Containers, and Code.

Container

Code

Cluster

Cloud

https://kubernetes.io/docs/concepts/security/overview/

How to Secure Your K8S Infrastructure

Cloud (Infrastructure)

Area of Concern for Kubernetes Infrastructure	Recommendation
Network access to API Server (Control plane)	All access to the Kubernetes control plane is not allowed publicly on the internet and is controlled by network access control lists restricted to the set of IP addresses needed to administer the cluster.
Network access to Nodes (nodes)	Nodes should be configured to only accept connections (via network access control lists)from the control plane on the specified ports, and accept connections for services in Kubernetes of type NodePort and LoadBalancer. If possible, these nodes should not be exposed on the public internet entirely.
Kubernetes access to Cloud Provider API	Each cloud provider needs to grant a different set of permissions to the Kubernetes control plane and nodes. It is best to provide the cluster with cloud provider access that follows the principle of least privilege for the resources it needs to administer. The Kops documentation provides information about IAM policies and roles.

SOURCE: https://kubernetes.io/docs/concepts/security/overview/; https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/

How to Secure Your K8S Infrastructure

Cloud (Infrastructure)

Area of Concern for Kubernetes Infrastructure	Recommendation
Access to etcd	Access to etcd (the datastore of Kubernetes) should be limited to the control plane only. Depending on your configuration, you should attempt to use etcd over TLS. More information can be found in the etcd documentation.
etcd Encryption	Wherever possible it's a good practice to encrypt all drives at rest, but since etcd holds the state of the entire cluster (including Secrets) its disk should especially be encrypted at rest.

SOURCE: https://kubernetes.io/docs/concepts/security/overview/; https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/

How to Secure Your K8S Cluster

Cluster

Area of Concern for Kubernetes Infrastructure	Recommendation
RBAC Authorization (Access to the Kubernetes API)	Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization. RBAC authorization uses the rbac.authorization.k8s.io API group to drive authorization decisions, allowing you to dynamically configure policies through the Kubernetes API.
Authentication	Users access the API using kubectl, client libraries, or by making REST requests. Both human users and Kubernetes service accounts can be authorized for API access. https://kubernetes.io/docs/reference/access-authn-authz/controlling-access/

SOURCE: https://kubernetes.io/docs/concepts/security/overview/; https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/ ; https://kubernetes.io/docs/reference/access-authn-authz/rbac/

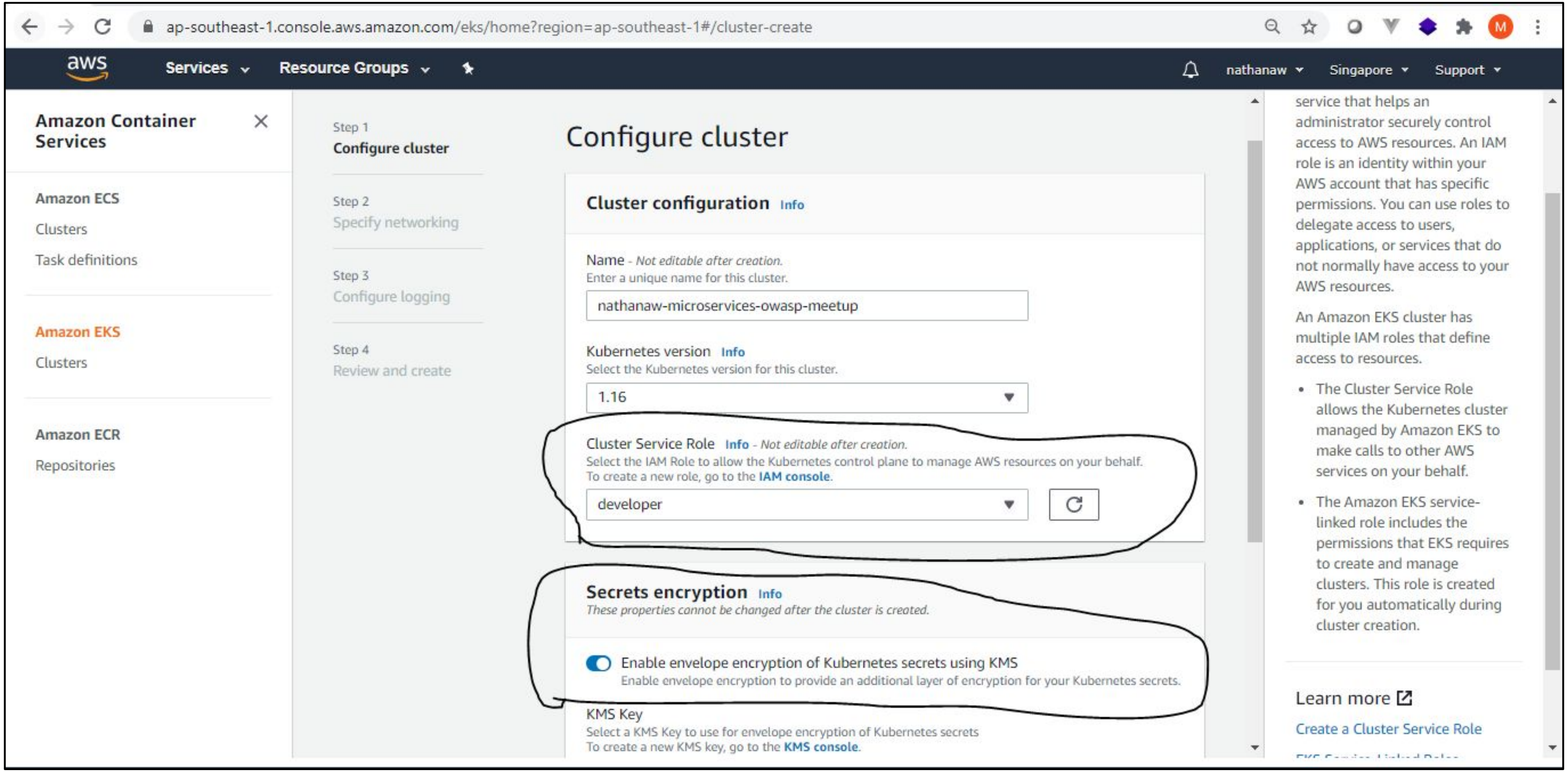
How to Secure Your K8S Cluster

Cluster

Area of Concern for Kubernetes Infrastructure	Recommendation
Application secrets management (and encrypting them in etcd at rest)	https://kubernetes.io/docs/concepts/configuration/secret/ https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/
Pod Security Policies	https://kubernetes.io/docs/concepts/policy/pod-security-policy/
Quality of Service (and Cluster resource management)	https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/
Network Policies	https://kubernetes.io/docs/concepts/services-networking/network-policies/
TLS For Kubernetes Ingress	https://kubernetes.io/docs/concepts/services-networking/ingress/#tls

SOURCE: https://kubernetes.io/docs/concepts/security/overview/; https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/

Sample AWS EKS Cluster Configuration



Configure cluster

Cluster configuration Info

Name - *Not editable after creation.*
Enter a unique name for this cluster.
nathanaw-microservices-owasp-meetup

Kubernetes version Info
Select the Kubernetes version for this cluster.
1.16

Cluster Service Role Info - *Not editable after creation.*
Select the IAM Role to allow the Kubernetes control plane to manage AWS resources on your behalf.
To create a new role, go to the IAM console.
developer

Secrets encryption Info

These properties cannot be changed after the cluster is created.

● Enable envelope encryption of Kubernetes secrets using KMS
Enable envelope encryption to provide an additional layer of encryption for your Kubernetes secrets.

KMS Key
Select a KMS Key to use for envelope encryption of Kubernetes secrets
To create a new KMS key, go to the KMS console.

service that helps an administrator securely control access to AWS resources. An IAM role is an identity within your AWS account that has specific permissions. You can use roles to delegate access to users, applications, or services that do not normally have access to your AWS resources.

An Amazon EKS cluster has multiple IAM roles that define access to resources.

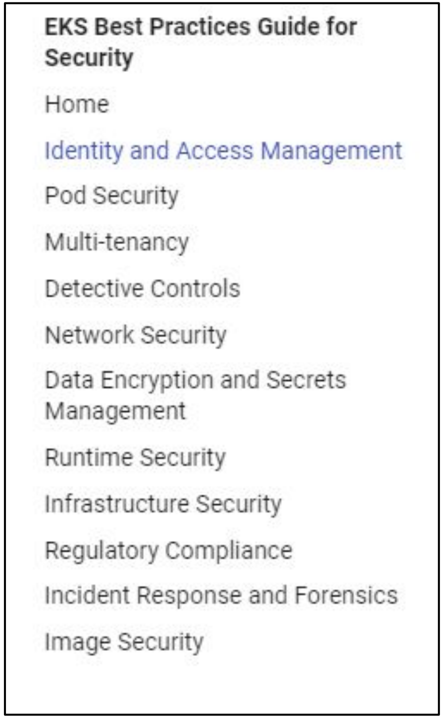
- The Cluster Service Role allows the Kubernetes cluster managed by Amazon EKS to make calls to other AWS services on your behalf.
- The Amazon EKS service-linked role includes the permissions that EKS requires to create and manage clusters. This role is created for you automatically during cluster creation.

Learn more

Create a Cluster Service Role

https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/

AWS EKS Security Best Practices



EKS Best Practices Guide for Security

Home

Identity and Access Management

Pod Security

Multi-tenancy

Detective Controls

Network Security

Data Encryption and Secrets Management

Runtime Security

Infrastructure Security

Regulatory Compliance

Incident Response and Forensics

Image Security

1. Controlling Access to EKS Clusters
2. Don't use a service account token for authentication
3. Employ least privileged access to AWS Resources
4. Use IAM Roles when multiple users need identical access to the cluster
5. Employ least privileged access when creating RoleBindings and ClusterRoleBindings
6. Make the EKS Cluster Endpoint private
7. Restrict the containers that can run as privileged
8. Do not run processes in containers as root
9. Never run Docker in Docker or mount the socket in the container
10. Create minimal images
11. And many more…

SOURCE: https://aws.github.io/aws-eks-best-practices/iam/

How to Secure Your K8S Container

Container

Area of Concern for Kubernetes Infrastructure	Recommendation
Container Vulnerability Scanning and OS Dependency Security	As part of an image build step, you should scan your containers for known vulnerabilities.
Image Signing and Enforcement	Sign container images to maintain a system of trust for the content of your containers.
Disallow privileged users	When constructing containers, consult your documentation for how to create users inside of the containers that have the least level of operating system privilege necessary in order to carry out the goal of the container
Restrict the containers that can run as privileged	As mentioned, containers that run as privileged inherit all of the Linux capabilities assigned to root on the host. Seldom do containers need these types of privileges to function properly. You can reject pods with containers configured to run as privileged by creating a pod security policy.

Container Runtime Security - Image Scanning

Compromised Images in Registry

Image scanning: The Docker security scanning process typically includes:

• Checking the software packages, binaries, libraries, operative system files and more against well known vulnerabilities databases. Some Docker scanning tools have a repository containing the scanning results for common Docker images. These tools can be used as a cache to speed up the process.

• Analyzing the Dockerfile and image metadata to detect security sensitive configurations like running as privileged (root) user, exposing insecure ports, using based images tagged with “latest” rather than specific versions for full traceability, user credentials, etc.

• User defined policies, or any set of requirements that you want to check for every image. This includes software packages blacklists, base images whitelists, whether a SUID file has been set, etc.

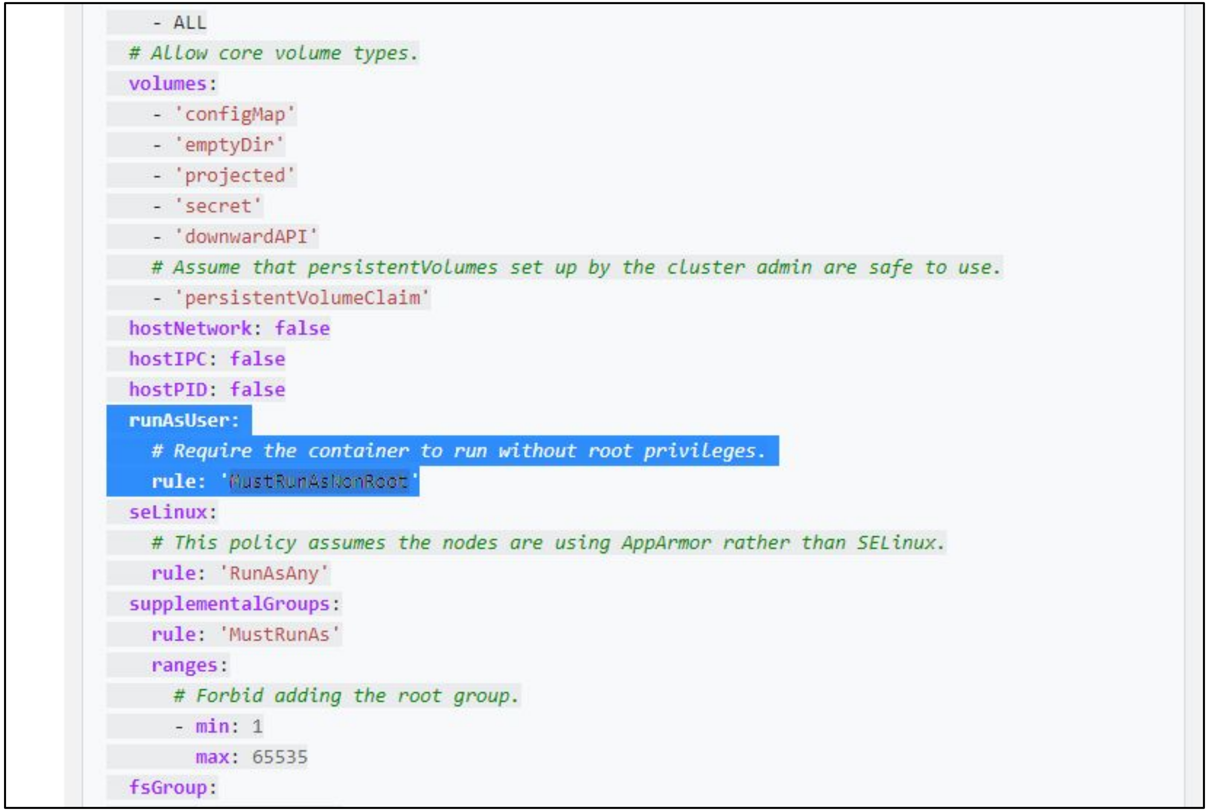
clair

A Container Image Security Analyzer by CoreOS

CoreOS/Clair: An open source project for the static analysis of vulnerabilities in application containers (currently including appc/Rkt and Docker).

Restrict the containers that can run as privileged - Rule:<u>MustRunAsNonRoot</u>

Container



```
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin are safe to use.
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    # Require the container to run without root privileges.
    rule: 'MustRunAsNonRoot'
  seLinux:
    # This policy assumes the nodes are using AppArmor rather than SELinux.
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  fsGroup:
```

https://kubernetes.io/docs/concepts/policy/pod-security-policy/#users-and-groups

How to Secure Your Application Code on K8S

Code

Access over TLS only

If your code needs to communicate by TCP, perform a TLS handshake with the client ahead of time. With the exception of a few cases, encrypt everything in transit. Going one step further, it's a good idea to encrypt network traffic between services. This can be done through a process known as mutual or mTLS which performs a two sided verification of communication between two certificate holding services.

Limiting port ranges of communication

This recommendation may be a bit self-explanatory, but wherever possible you should only expose the ports on your service that are absolutely essential for communication or metric gathering.

Static Code Analysis

Most languages provide a way for a snippet of code to be analyzed for any potentially unsafe coding practices. Whenever possible you should perform checks using automated tooling that can scan codebases for common security errors. Some of the tools can be found at: https://owasp.org/www-community/Source_Code_Analysis_Tools

Practice Writing Secure By Design Code!

SOURCE: https://kubernetes.io/docs/concepts/security/overview/

Service Mesh - Definition

“A service mesh, like the open source project Istio, is a way to control how different parts of an application share data with one another. Unlike other systems for managing this communication, a service mesh is a dedicated infrastructure layer built right into an app.” - Red Hat

“A service mesh is a configurable, low‑latency infrastructure layer designed to handle a high volume of network‑based interprocess communication among application infrastructure services using application programming interfaces (APIs).” - Nginx

Service Mesh To Help Improve Security Posture

Traffic observability that Service mesh offers, combined with external traffic profiling and analysis tools, enables security-related traffic auditing and monitoring for detection and investigation of network behavior anomalies.

Service mesh traffic can be automatically encrypted with mutual endpoint authentication, using mTLS.

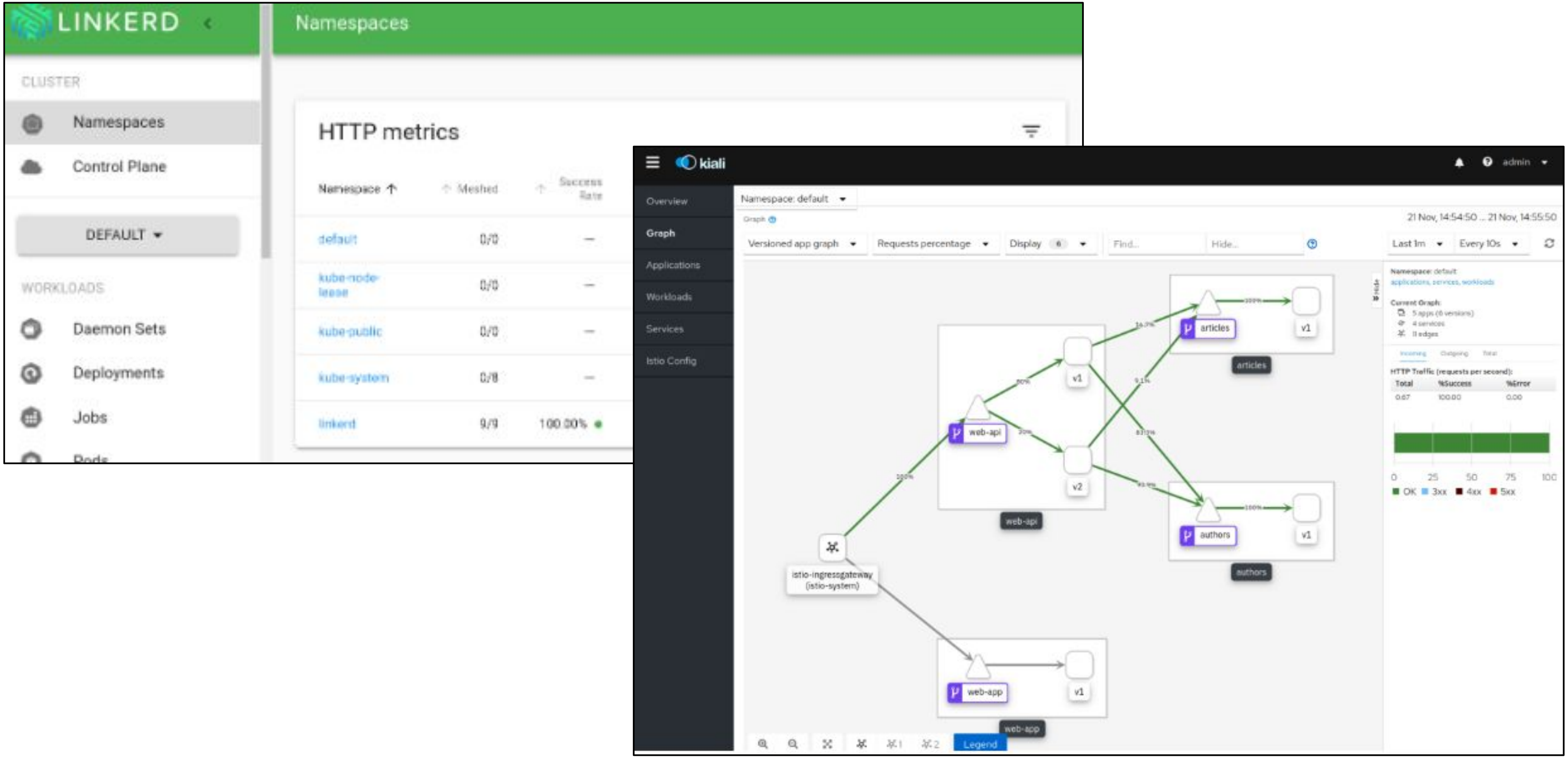
Fine-grained role-based access control at the application layer network protocol can be used for micro-segmentation, further enhancing users' abilities to limit which services interact and in what ways.

Authenticates workloads' identities and issues and manages certificates for them used in creating the mesh connectivity.

Configurable authentication policies and secure naming information ensure traffic authorization at the transport layer.

SOURCE: https://www.alcide.io/service-mesh-security/

Service Mesh - Linkerd and Istio



HTTP metrics

Namespace ↑	Meshed	Success Rate
default	0/0	—
kube-node-lease	0/0	—
kube-public	0/0	—
kube-system	0/8	—
linkerd	9/9	100.00% ●

Namespace: default

21 Nov, 14:54:50 ... 21 Nov, 14:55:50

Versioned app graph · Requests percentage · Display 6 · Find... · Hide... · Last 1m · Every 10s

Namespace: default
applications, services, workloads

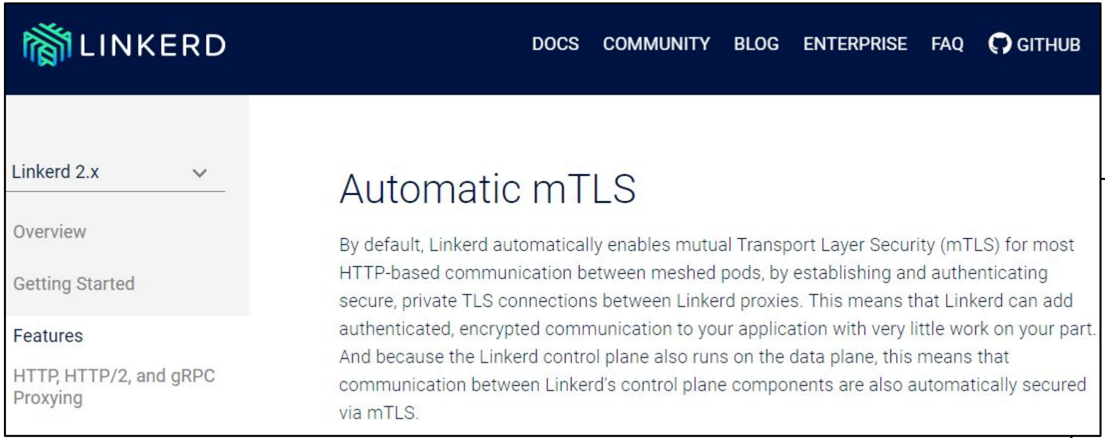
Current Graph:
5 apps (6 versions)
4 services
11 edges

Incoming · Outgoing · Total

HTTP Traffic (requests per second):

Total	%Success	%Error
0.67	100.00	0.00

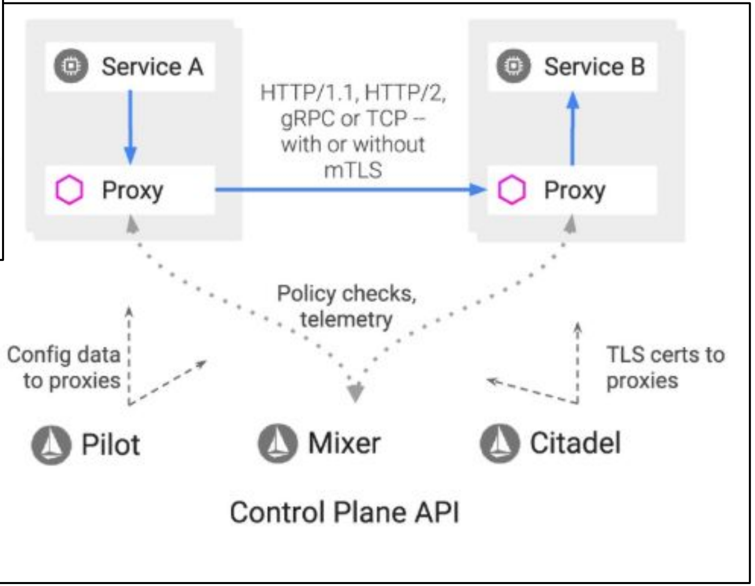
Service Mesh - Automatic mTLS



Automatic mTLS

By default, Linkerd automatically enables mutual Transport Layer Security (mTLS) for most HTTP-based communication between meshed pods, by establishing and authenticating secure, private TLS connections between Linkerd proxies. This means that Linkerd can add authenticated, encrypted communication to your application with very little work on your part. And because the Linkerd control plane also runs on the data plane, this means that communication between Linkerd's control plane components are also automatically secured via mTLS.

SOURCE: https://linkerd.io/

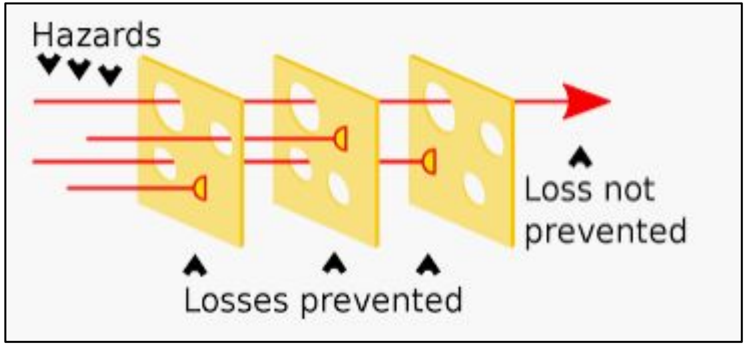


SOURCE: https://istio.io/

Service Mesh is not a panacea nor silver bullet to all the potential security ills and pitfalls. Vigilance and Defense-in-Depth Approach is still needed!

What’s Next + Final Words

- **Multi Cloud Reality - K8S Clusters spanning across multi-cloud**



The Swiss Cheese Model / Defense-in-Depth Approach Sorely Needed - No one size fits all

- “Know all your assets, well. Know them well. (especially all the component in the asset. E.g., the ETCD in K8S, Golang) and secure em’ all!
- “Secure by Design” Application: **Secure code is the best code**. Secure by design means that security is baked into your software design from the beginning.

Feel reach out to me @ https://www.linkedin.com/in/awnathan

- Currently an **AppDevSec** Digital Solutions Architect and a Full-Stack Developer in the Financial Services Industry (FSI)
    - First a Full-Stack Developer, then a Solutions Architect
    - Previously worked in a local bank as a Full-Stack Blockchain Engineer
    - *Have Designed, Built, Deployed and Operated **> 58** Unique Polyglot Based Production Grade Microservices (Micro Frontends, Backend for Frontends, Backends) over last 3 years*
- **Specialties** around API, Microservices that enables a Seamless & Frictionless Customer Journey Experience (CJX)
    - On “Hybrid-Multi” Cloud Native Platforms
    - On API, Microservices Security, Container Runtime Security and MITRE ATT&CK® for Kubernetes(K8S)
- Technology Stack: Golang, React, Kafka, Spring Boot, NodeJS, Apigee, Kong, Zuul, GraphQL, Azure Kubernetes Service (AKS), Elastic Kubernetes Service (EKS), Openshift, Service Mesh (Istio, Linkerd), Cloud Foundry and many more…
- Building Scalable, Secure and Robust APIs and Microservices is my passion!
- https://www.linkedin.com/in/awnathan
- ***Opinions/views expressed in the talk are solely my own and do not express the views or opinions of my employer.***

References/Sources

Oreilly - Container Security: Fundamental Technology Concepts that Protect Containerized Applications

https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf

https://skyao.gitbooks.io/microservice-collection/content/master/Andreas-Schroeder/static/microservice-architectures.pdf

https://owasp.org/www-chapter-singapore/assets/presos/Securing_your_APIs_-_OWASP_API_Top_10_2019,_Real-life_Case.pdf

https://aws.github.io/aws-eks-best-practices/iam/

References/Sources

https://developer.okta.com/blog/2020/03/23/microservice-security-patterns

https://www.infoq.com/podcasts/web-security-hack-anatomy/