KubeCon | CloudNativeCon
Europe 2020
Virtual

Threat Modelling: Securing Kubernetes Infrastructure & Deployments

Rowan Baker
@controlplaneio

controlplane

Acknowledgement

What this talk is about

- Threat Modelling Kubernetes
- Defining Kubernetes Security Controls & Architectures
- Testing
- SOC integration
- Addressing Compliance Culture Shock
- Gotchas

Threat Modelling in a Slide (ish)

KubeCon CloudNativeCon Europe 2020 Virtual

- What
- Why
- When
- Who
- Where
- How?

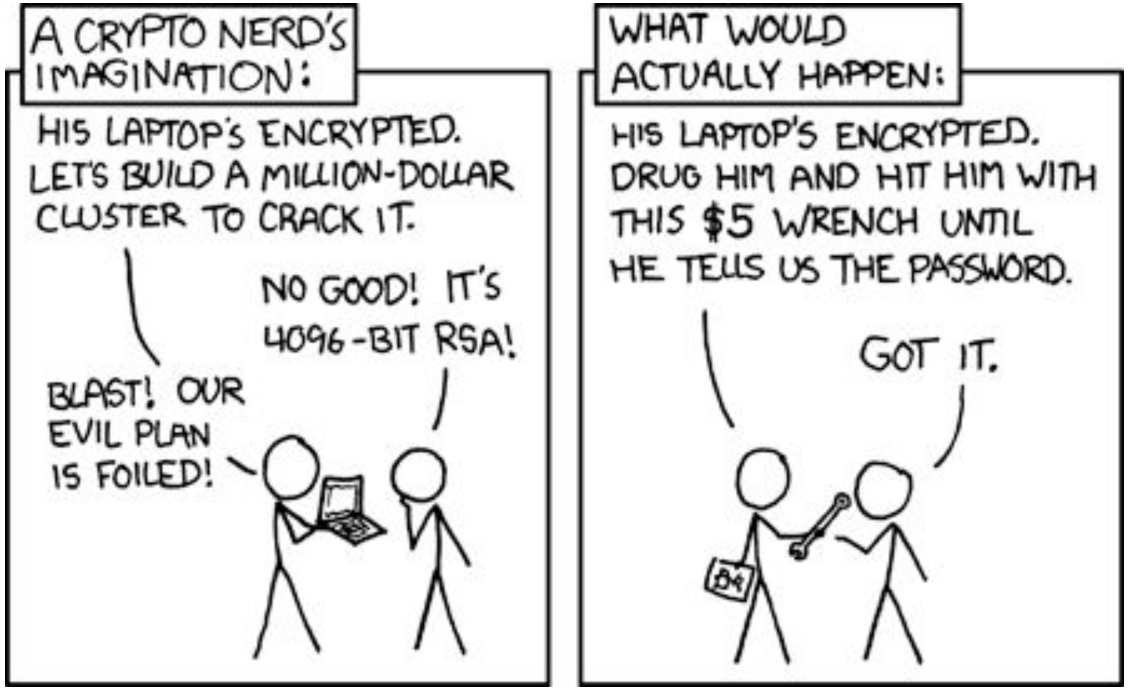
controlplane

Threat Modelling in a Slide (ish)

- What
- Why
- When
- Who
- Where
- How?

Used as both a noun and a verb

The exact definition doesn't matter, doing it does.



controlplane

Threat Modelling in a Slide (ish)

- What
- Why
- When
- Who
- Where
- How?

Threat modelling can prevent you from finding out about security issues when it's too late…

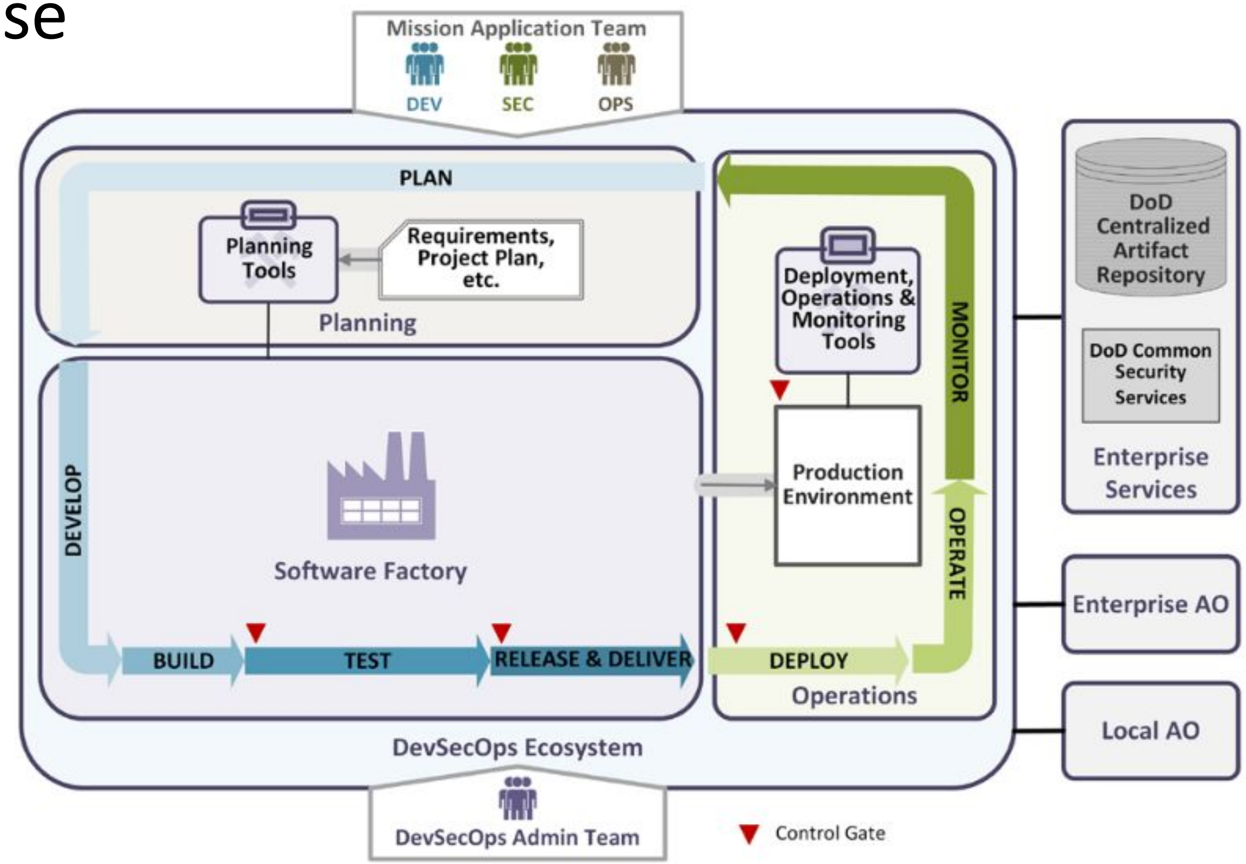
Threat Modelling in a Slide (ish)

KubeCon CloudNativeCon Europe 2020 Virtual

- What
- Why
- When
- Who
- Where
- How?

As early as possible

- Once a shared understanding is established
- When features are designed for every subsequent release



controlplane

Threat Modelling in a Slide (ish)

- What
- Why
- When
- **Who**
- Where
- How?

Each stakeholder brings their own unique perspective

Threat Modelling in a Slide (ish)

KubeCon | CloudNativeCon Europe 2020 Virtual

- What
- Why
- When
- **Who**
- Where
- How?

Architects know how things should work

controlplane

Threat Modelling in a Slide (ish)

KubeCon CloudNativeCon Europe 2020 Virtual

- What
- Why
- When
- **Who**
- Where
- How?

DevOps know how things *actually* work

controlplane

Threat Modelling in a Slide (ish)

- What
- Why
- When
- Who
- Where
- How?

And others:

- SOC/ VA/ Threat Intelligence
- Product Owners

Caution- if these groups are silo'd - run preparatory sessions.

Threat Modelling in a Slide (ish)

- What
- Why
- When
- Who
- Where
- How?

In a room with a whiteboard

Or

Over video conferencing tools

- At the mercy of collaborative tooling

controlplane

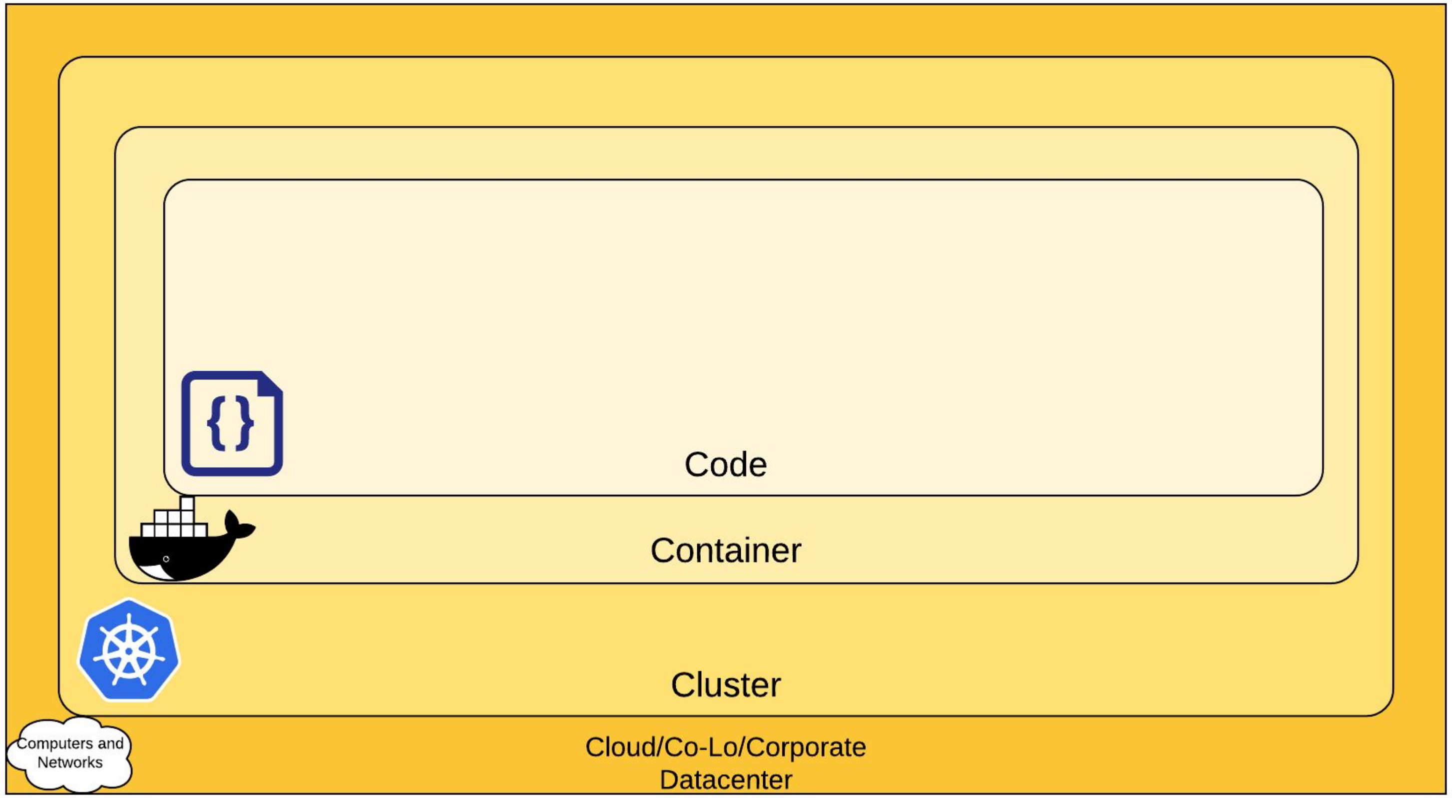
Threat Modelling in a Slide (ish)

- What
- Why
- When
- Who
- Where
- How?

4 steps:

1. What are you building?
2. What can go wrong once it’s built?
3. What should you do about those things that can go wrong?
4. Did you do a decent job of analysis?

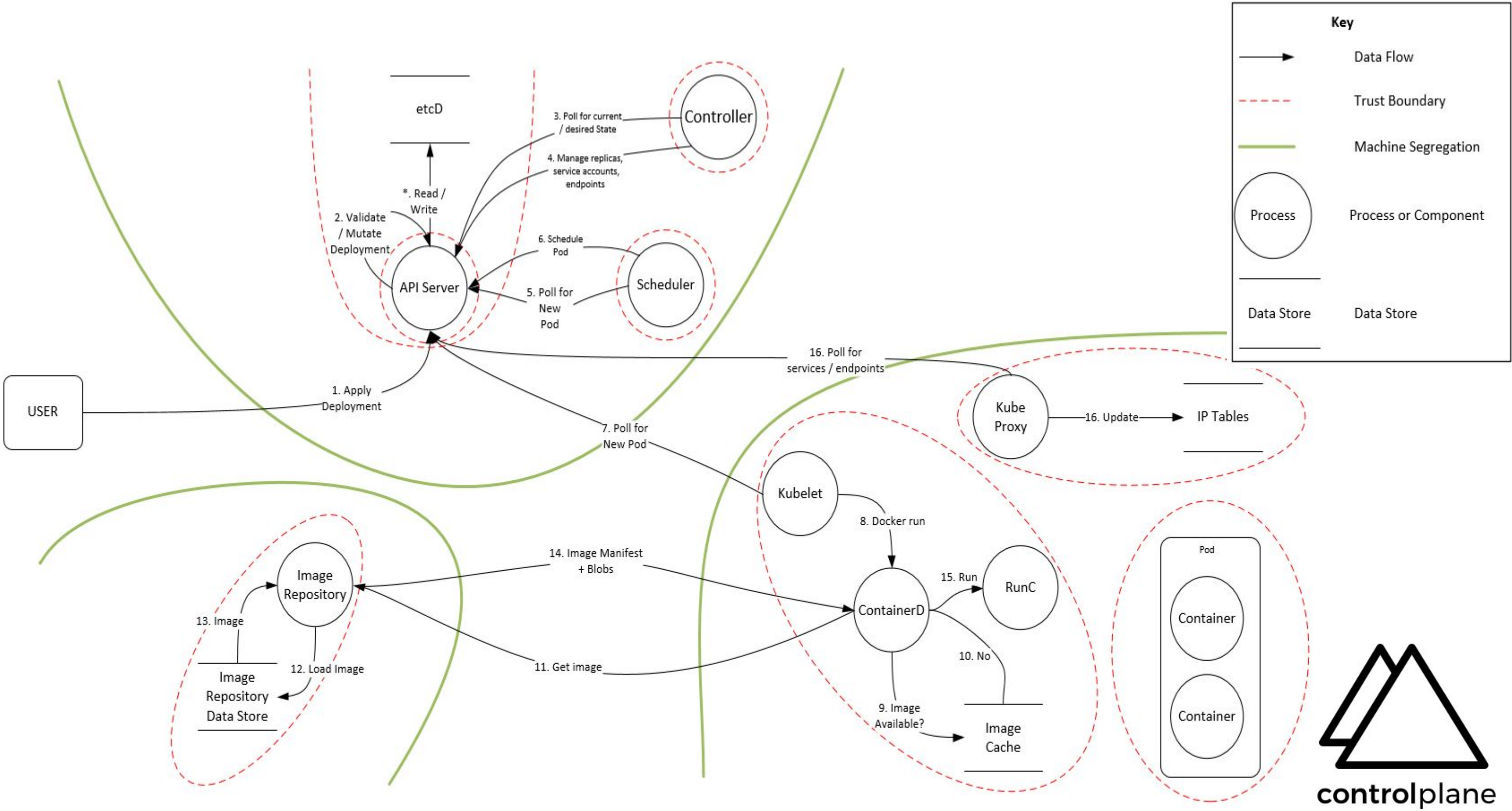
What does this look like for Kubernetes?



Kubernetes Cluster Threat Models

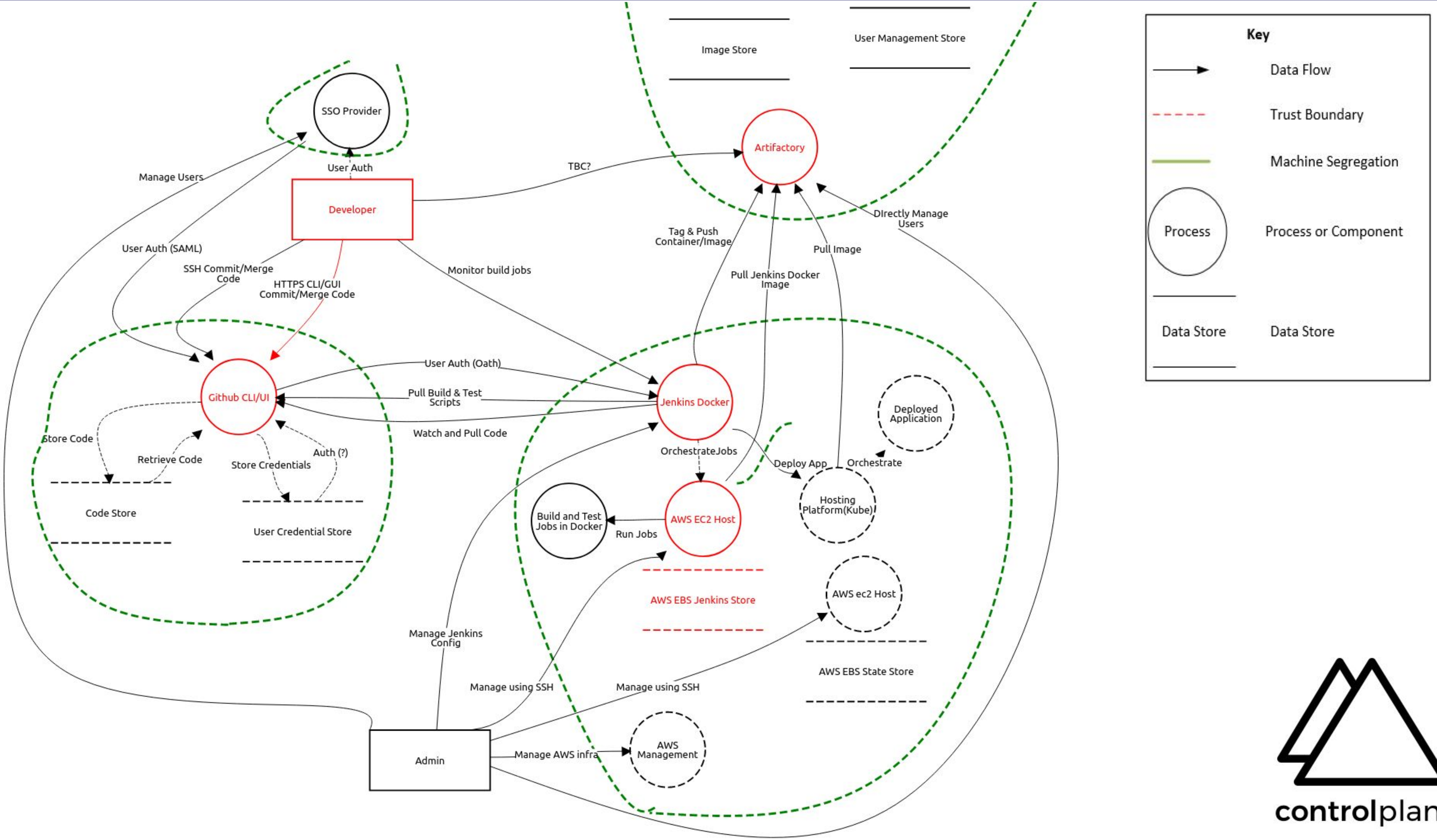
- Provisioning and Scaling
- Runtime & Cluster configuration
- CI/CD & Application deployment

Data Flow Diagram - Kubernetes Pod Launch



Data Flow Diagram - CI/CD

KubeCon CloudNativeCon Europe 2020 Virtual



controlplane

What can go wrong?

Techniques

- STRIDE
- PASTA

Sources

- MITRE ATT&CK
- Reverse engineer benchmarks

Brainstorm and make notes first

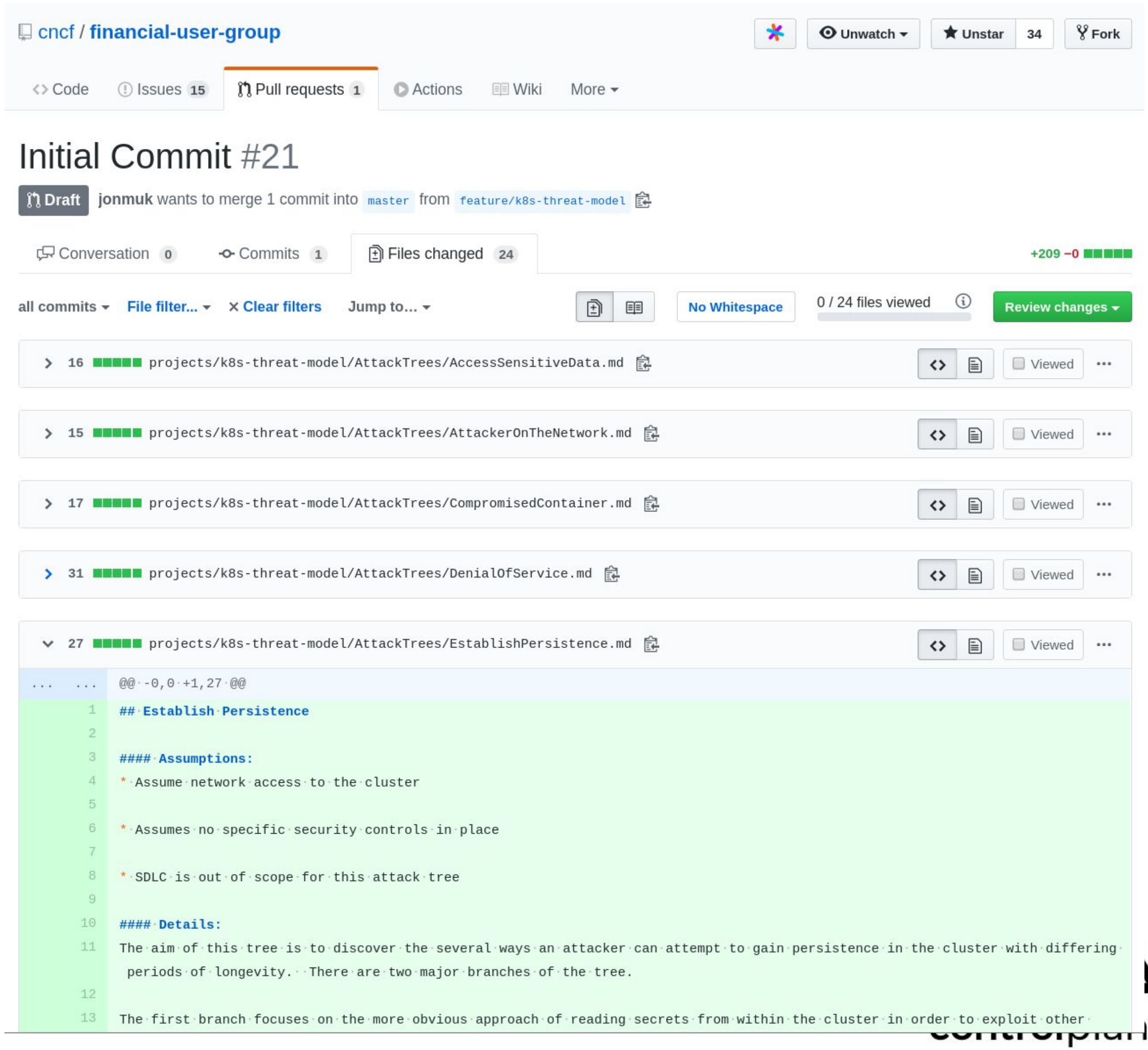
Element	S	T	R	I	D	E
External Entity	X		X			
Process	X	X	X	X	X	X
Data Flow		X		X	X	
Data Store		X	?	X	X	

Existing Runtime Models - CNCF Attack Trees

We worked together with other members of the CNCF Financial User Group to threat model the whole Kubernetes system

The initial set of Attack Trees are now open sourced and available on GitHub:

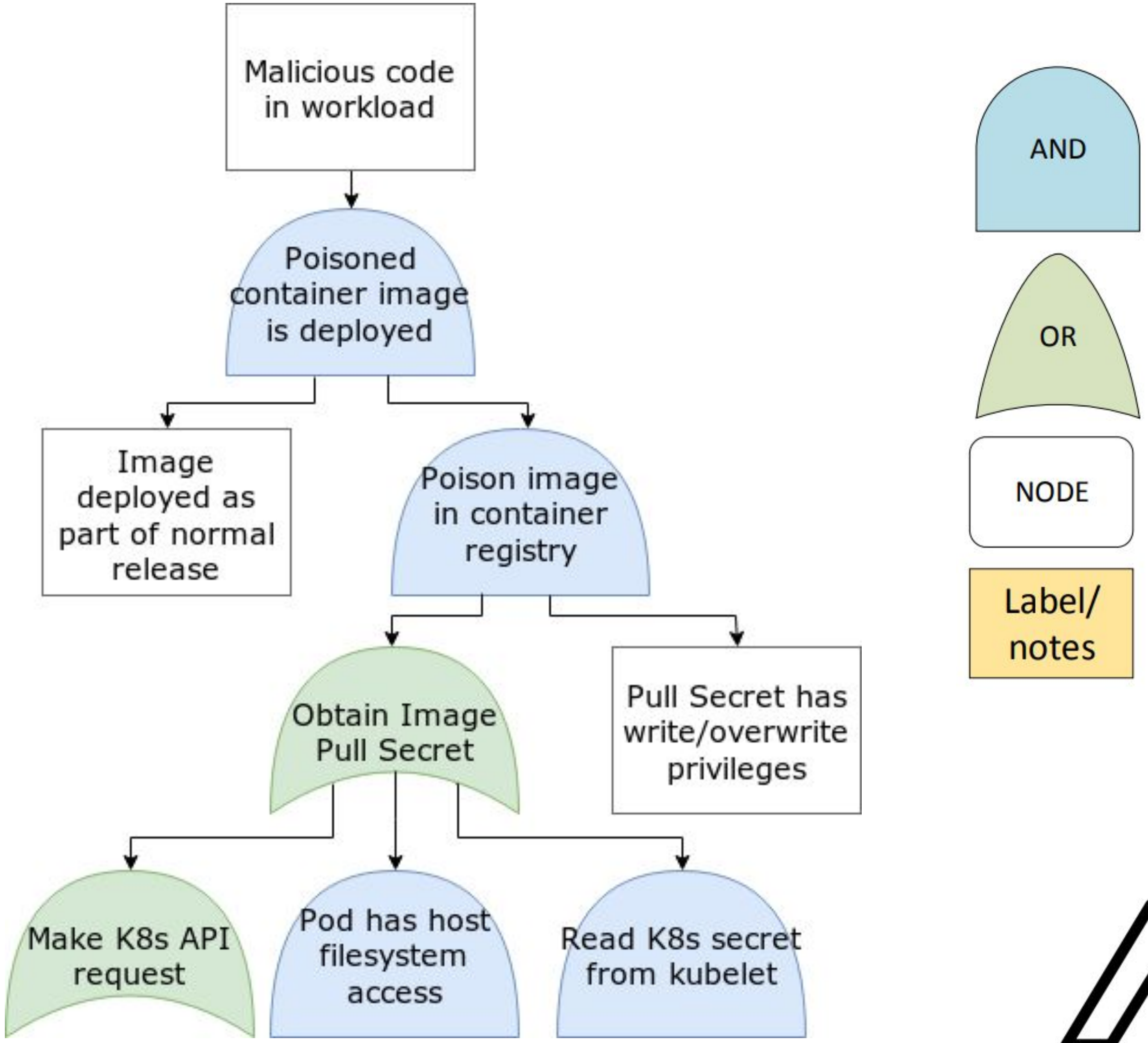
https://github.com/cncf/financial-user-group/tree/master/projects/k8s-threat-model



Attack Trees

"Attack trees provide a ***formal, methodical*** *way of* ***describing the security of systems,*** *based on varying attacks. Basically, you represent attacks against a system in a* ***tree structure****, with the* ***goal as the root node*** *and different ways of* ***achieving that goal as leaf nodes****."*

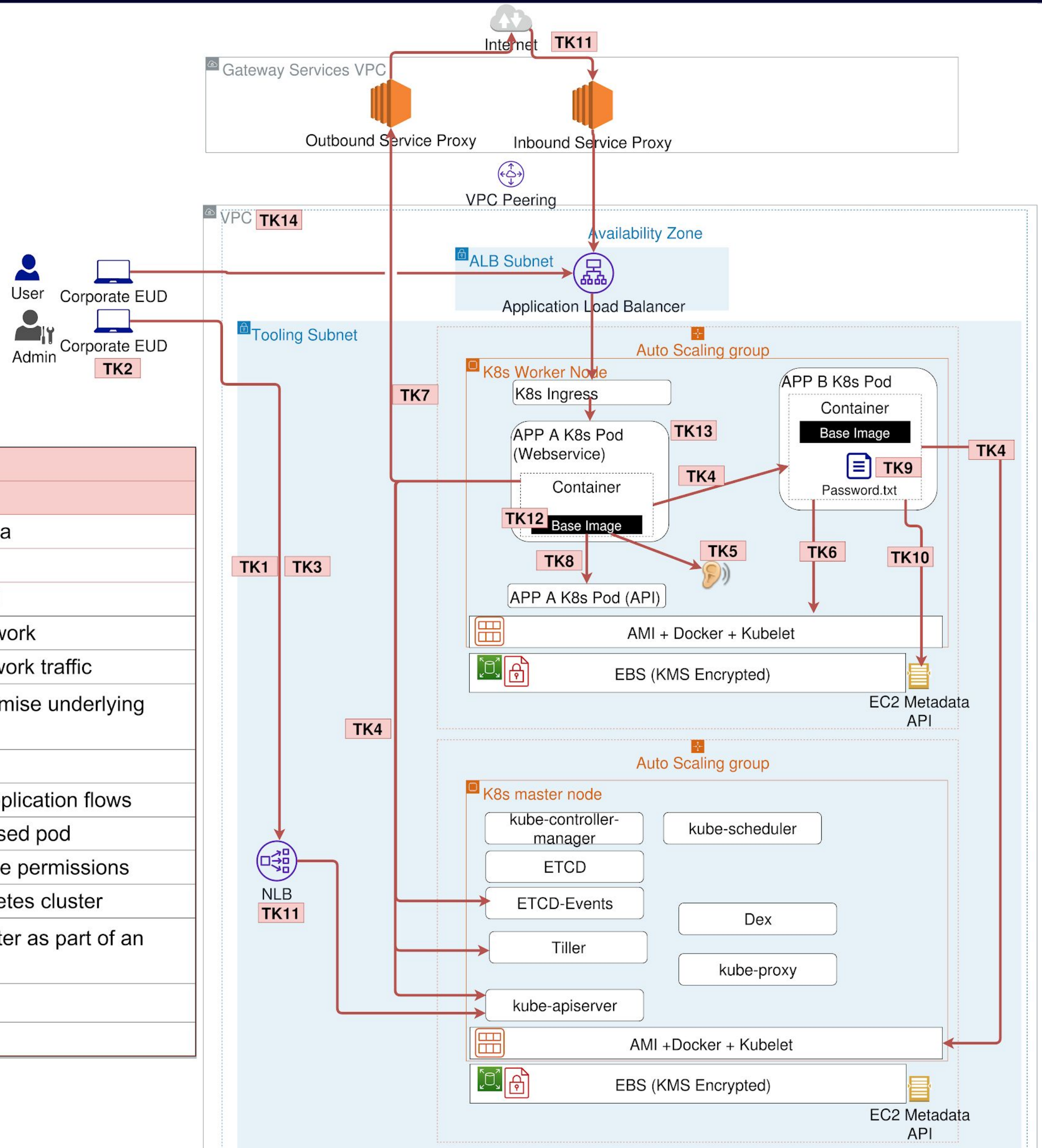
Bruce Schneier (1999)



Kubernetes Runtime - What can go wrong?

KubeCon | CloudNativeCon Europe 2020 Virtual

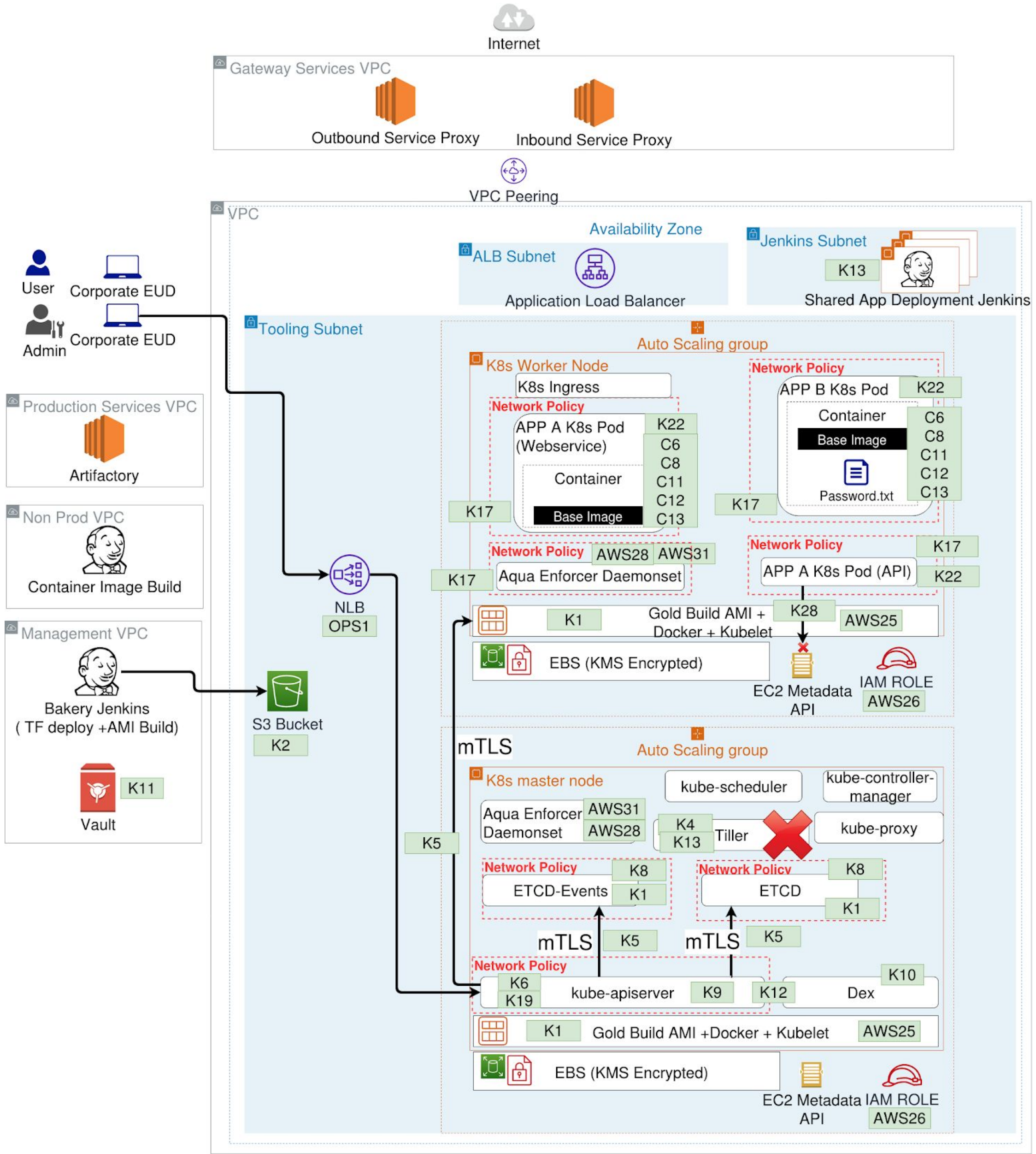
Threats	
ID	**Description**
TK1	Insider uses the K8s API to access application data
TK2	K8s API user/admin credentials are compromised
TK3	Application or cluster is misconfigured via K8s API
TK4	Compromised application Pod pivots over the network
TK5	Compromised application pod eavesdrops on network traffic
TK6	Compromised application pod attempts to compromise underlying node
TK7	Compromised Pod forms outbound connection
TK8	Compromised Pod harnesses/modifies existing application flows
TK9	Long-lived secrets are extracted from a compromised pod
TK10	Compromised Pod attempts to inherit EC2 instance permissions
TK11	External attacker attempts to compromise Kubernetes cluster
TK12	Malicious or Vulnerable code deployed within cluster as part of an application or the Kubernetes platform
TK13	Autoscaling application DDOS of K8s Cluster
TK14	Failure of Platform or AWS Region



controlplane

What are we going to do about the things that go wrong?

KubeCon CloudNativeCon Europe 2020 Virtual

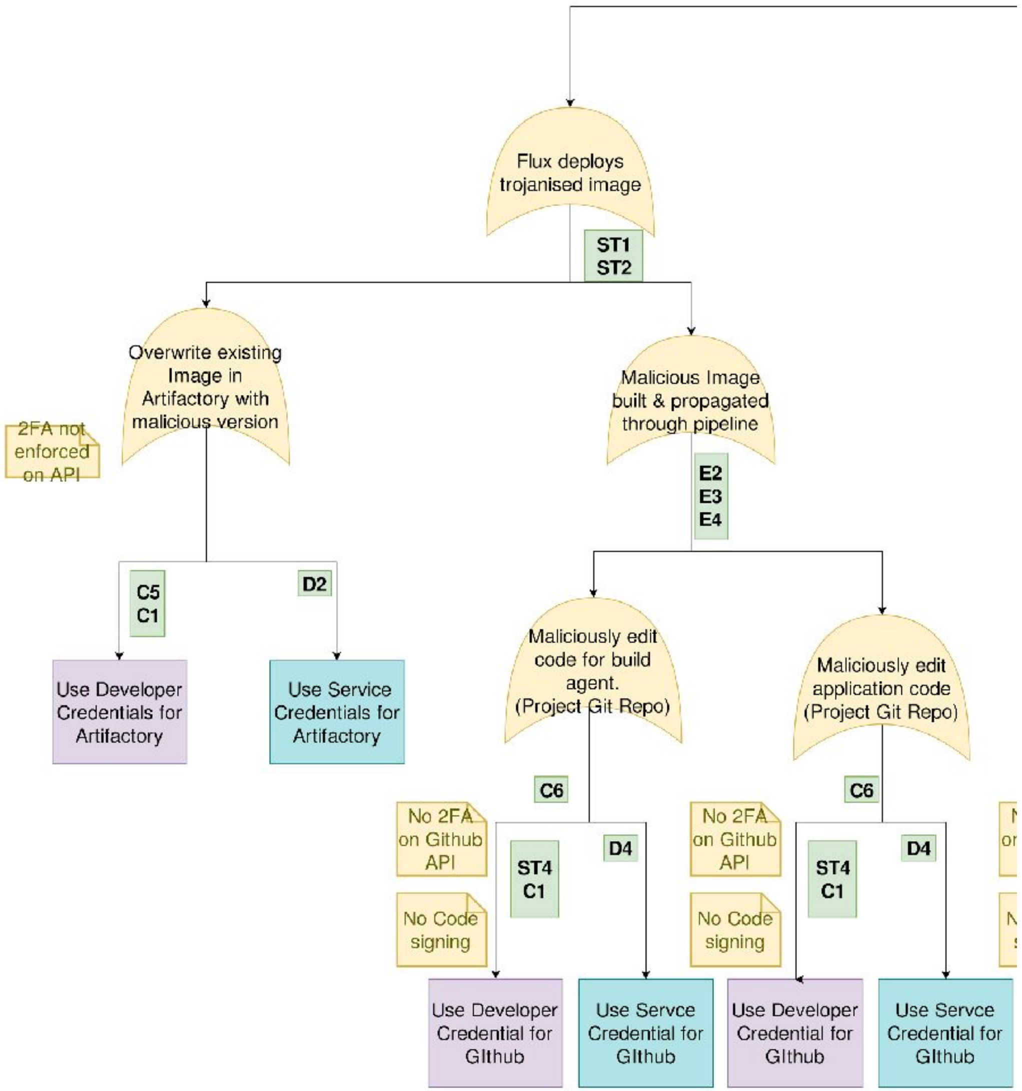


New Security Controls	
REQ ID	**Description**
OPS1	Use of Dedicated Devices & Networks for Management (Whitelist specific Offices)
AWS 25	Harden EC2 instances
AWS 26	Restrict EC2 Instance IAM Roles (Segregate and minimise permissions)
AWS 28	Network based IDS/IPS (Aqua Enforcer Daemonset)
AWS 31	Container based IDS/IPS (Aqua Enforcer Daemonset)
K1	CIS Benchmark for Kubernetes (etcd Encryption Provider & harden host file permissions))
K2	Deploy K8s using IaC (create backup strategy)
K4	GitOps deployment
K5	Enforce control plane and etcd mTLS
K6	Segregated and Firewalled Kubernetes control plane (Using K8s Network Policy)
K8	Firewalled etcd cluster (Using K8s Network Policy)
K9	Kubernetes - Logging and protective monitoring (Enable audit logs)
K10	Federate auth to 3rd party Identity Provider and enforce 2FA (enforce 2FA)
K11	Cluster Admin role must be a breakglass role (store in Vault for breakglass)
K12	Enforce User RBAC and Least Privilege (Create Roles)
K13	Enforce Service Account RBAC & Least Privilege (remove Tiller + dedicated service account for Jenkins deployment)
K17	Network Policy
K19	Enable & utilise Admission Controllers (Enable Node Restriction Controller)
K22	Pods deployed in compliance with Pod Security Policies
K28	AWS EC2 API Metadata Concealment / Restriction (Using Network Policy)
C6	Use of Minimal Base Images
C8	Container Image/Dependency scanning for CVEs (prevent vulnerable images from being pulled)
C11	Immutability of running containers
C12	Hardened Container Security Contexts and Resource Allocation
C13	Non-root user container process ownership

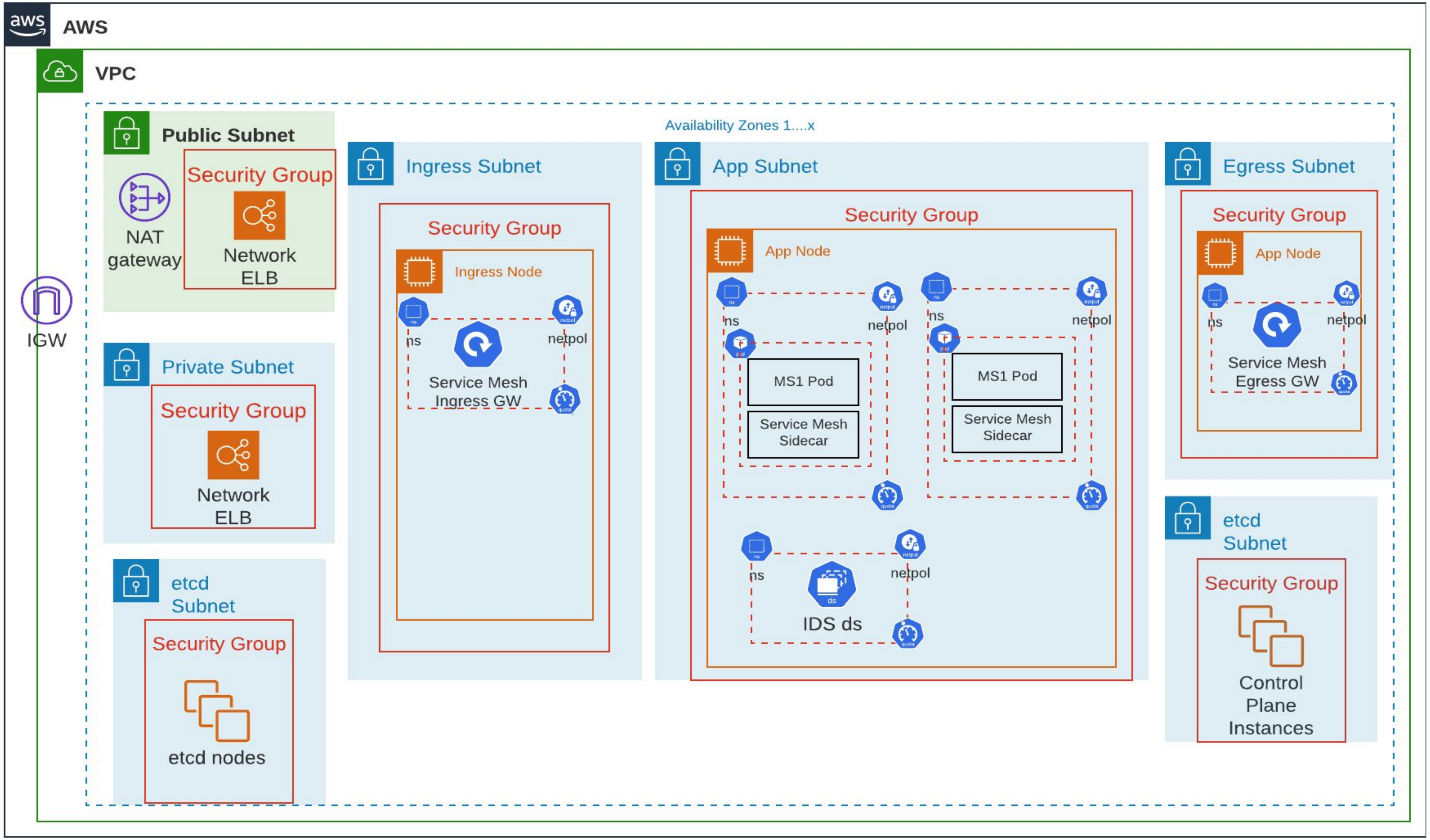
controlplane

What are we going to do about the things that go wrong?

Security Requirements	
ID	**Description**
A	**Operational & EUD requirements**
A1	Developer awareness training
A2	Use of Organisation approved EUDS
A3	Use of encrypted Keychain/password manager
A4	Patching
A5	Logging & Monitoring
A6	Audit
A7	Production pull request review process
A8	Use a tool to show differences between running state and version controlled configuration
B	**Hardening, Encryption & Networking**
B1	Encryption in transit
B2	Encryption at rest
B3	Port & Service minimisation
B4	Use of IP Whitelisting/ VPN
B5	K8s hardened in accordance with best practice (e.g. CIS benchmark)
B6	CI server (Jenkins) hardened in ccordance with best practice
B7	Disable CI server (Jenkins) script console
C	**Developer RBAC, least privilege & segregation of duties**
C1	Use of 2FA
C2	Developer permissions in CI server restricted to read & run jobs. CI server managed by Cloud Engineering
C3	No Developer access to K8s staging or production cluster
C4	Developers don't have deploy permissions or access to Flux within the K8s dev cluster
C5	Developer permissions in Artifactory are read only
C6	Protected Branches enforcing peer review, code ownership & includes administrators.
C8	Developer permissions on Github Flux config repos restricted to read only
C9	Developer credentials on X-ray limited to "View Watches"
D	**CI servers' service accounts least privilege**
D1	CI server permissions within K8s restricted to reading Flux logs- no deployment permissions
D2	CI server permissions within Artifactory



Complementing Controls - Networking



Complementing Controls - Runtime

Security Context for Pods & Containers

- Run as non-root User
- Run as unprivileged
- Drop all Linux capabilities
- Use AppArmor Profiles/ SELinux

Container Based IDS

Sandbox technologies


```yaml
securityContext:
  runAsUser: 1000
  runAsGroup: 3000
  fsGroup: 2000
volumes:
- name: sec-ctx-vol
  emptyDir: {}
containers:
- name: securecontainer
  image: busybox
  command: [ "sh", "-c", "sleep 1h" ]
  volumeMounts:
  - name: sec-ctx-vol
    mountPath: /data/demo
  securityContext:
    allowPrivilegeEscalation: false
    runAsNonRoot: true
    readOnlyRootFilesystem: true
    capabilities:
      drop:
      - All
```

Complementing Controls - Runtime

Security Context for Pods & Containers

- Run as non-root User
- Run as unprivileged
- Drop all Linux capabilities
- Use AppArmor Profiles/ SELinux

Container Based IDS

Sandbox technologies

Falco

controlplane

Complementing Controls - Runtime

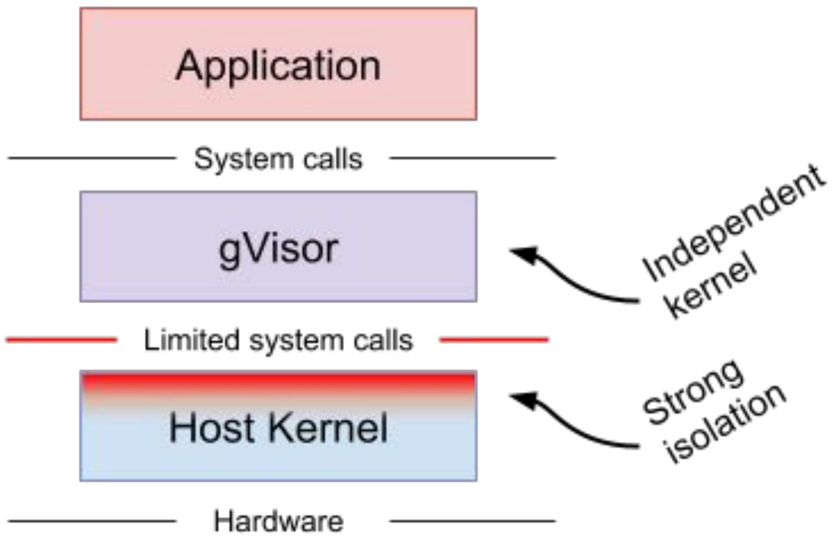
KubeCon CloudNativeCon Europe 2020 Virtual

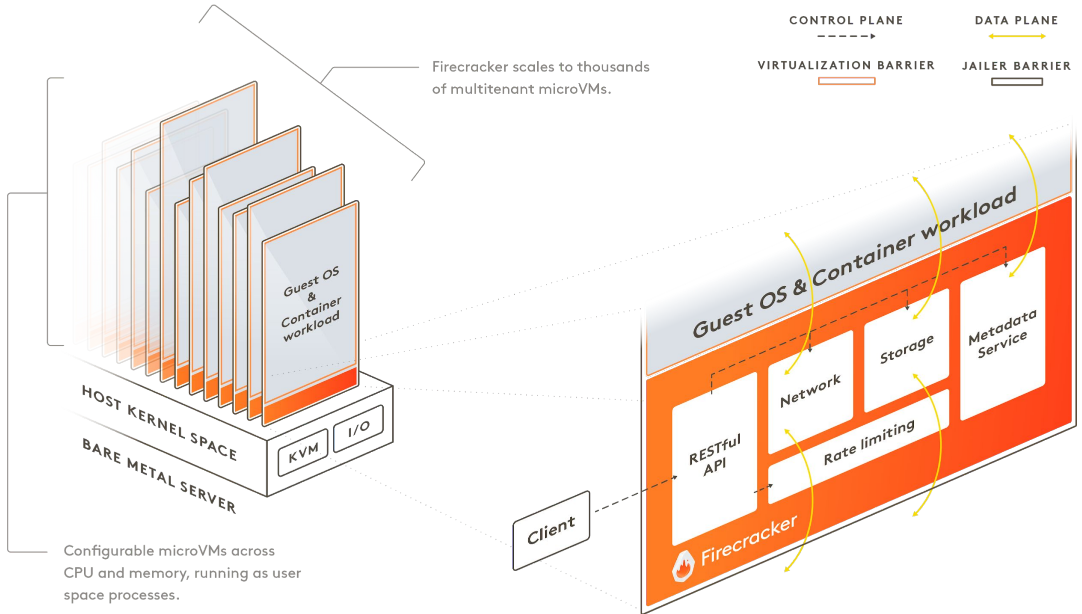
Security Context for Pods & Containers

- Run as non-root User
- Run as unprivileged
- Drop all Linux capabilities
- Use AppArmor Profiles/ SELinux

Container Based IDS

Sandbox technologies

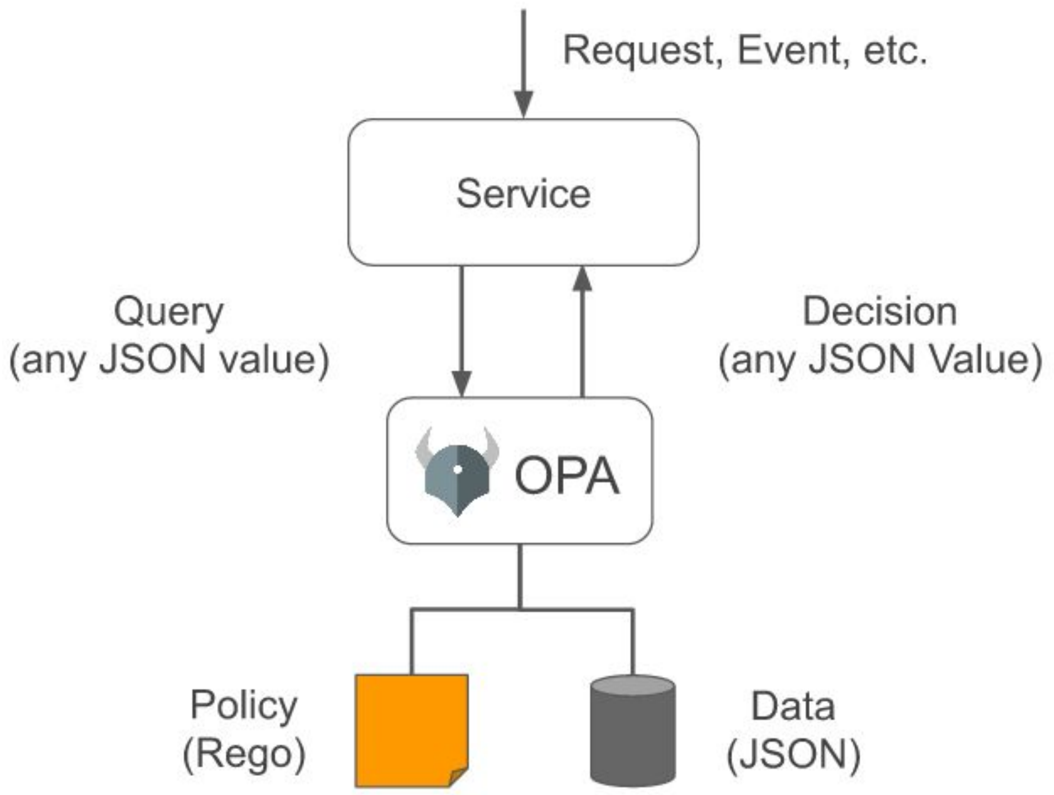




controlplane

Complementing Controls - RBAC & Policy

- Kubernetes RBAC
- Admission Controllers
- Open Policy Agent
  - Custom Policy
  - Pod Security Policy
  - Multiple Implementations
    - Gatekeeper
    - Plain OPA



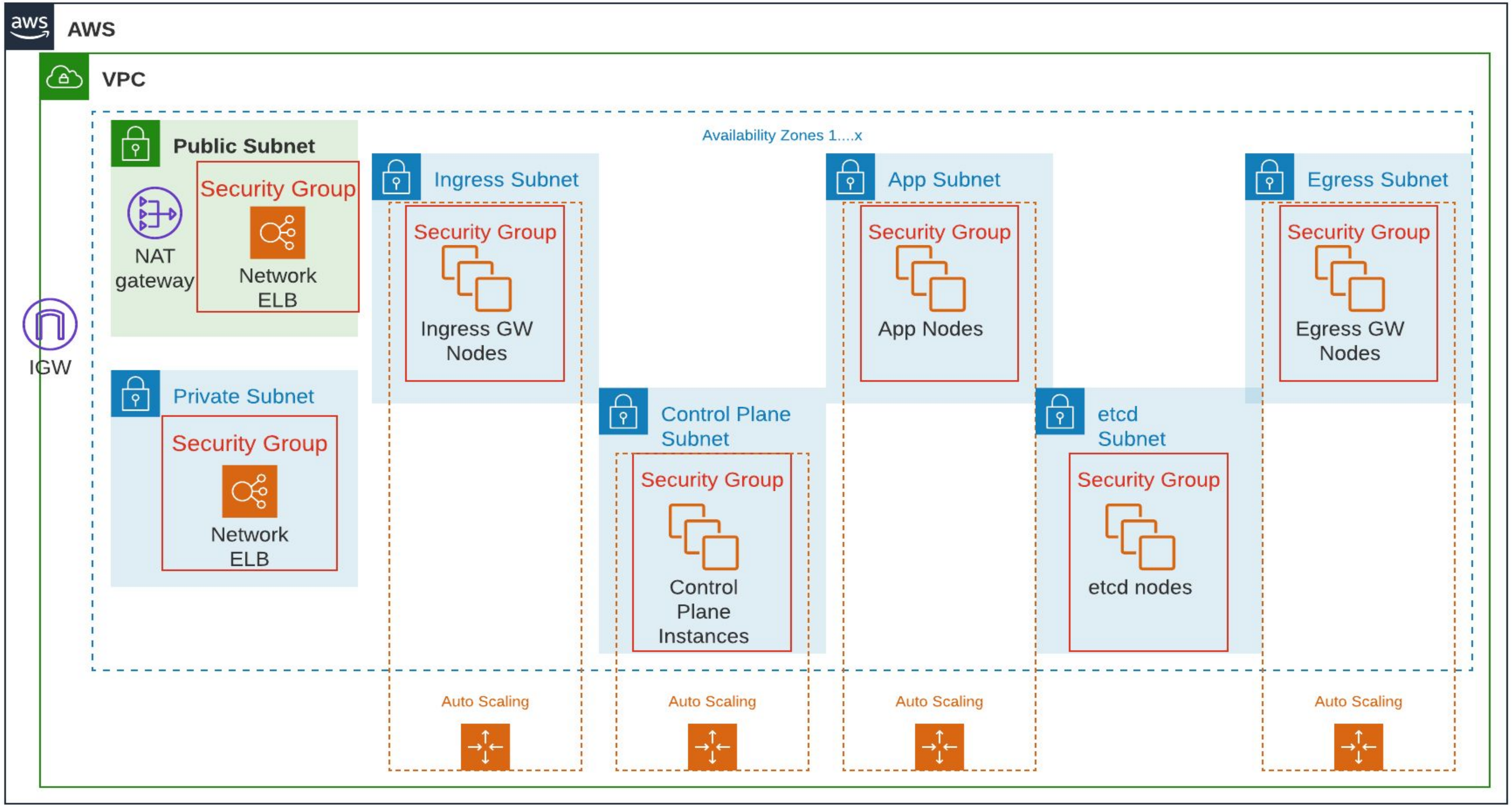
Policy Decoupling

Complementing Controls - Supply Chain Security

Base image	Code	Build	Application image	Deploy
Images: Docker Distribution (Hub)	**Updates:** TUF, Notary	**Pipeline metadata:** Grafeas, in-toto	**Vulnerability scanning:** Clair, Micro Scanner, Anchore Open Source Engine	**Admission control:** K8s admission controllers, Kritis, Portieris

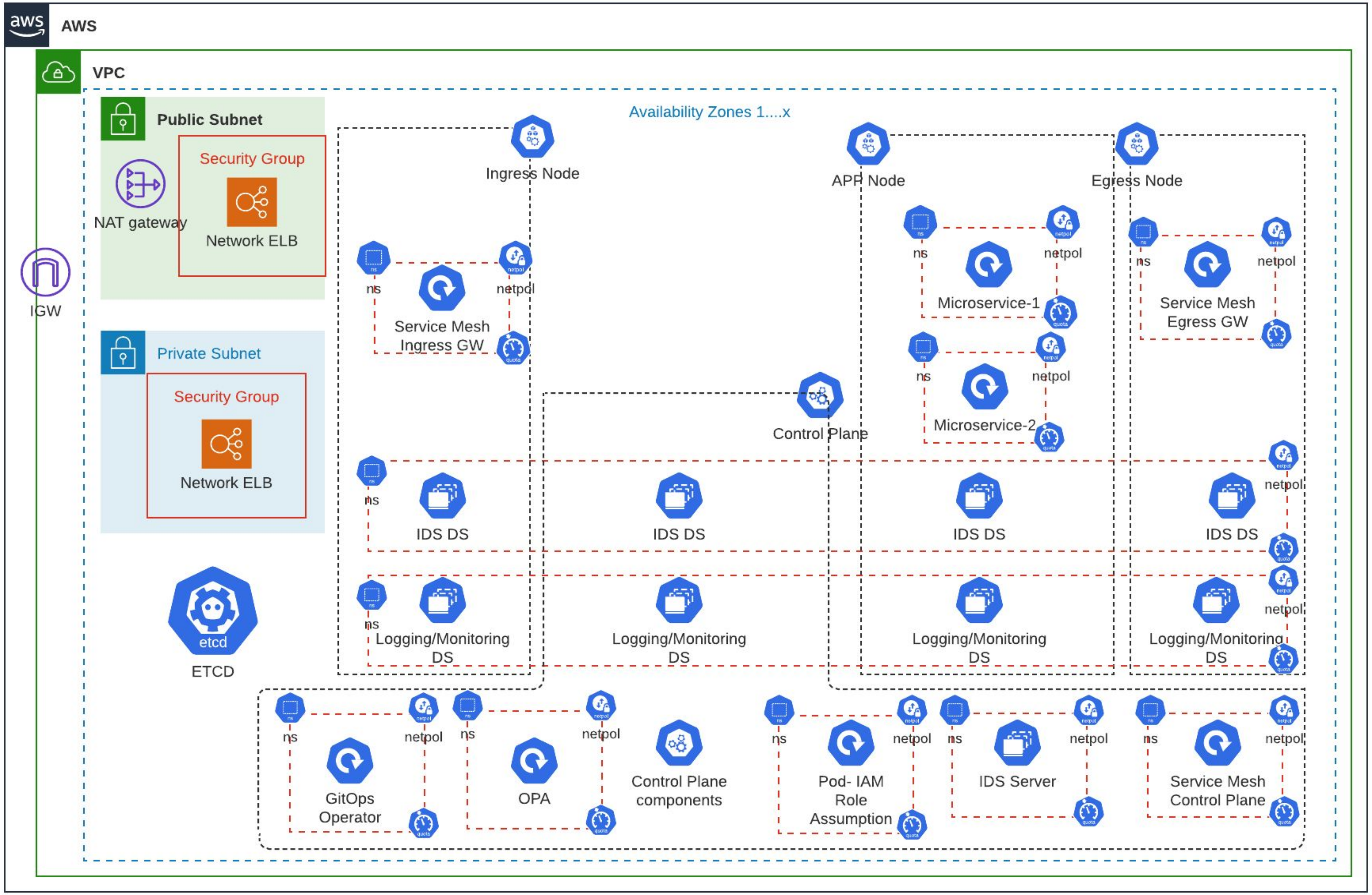
(DevSecOps Kubernetes Pipeline Workshop KubeCon Seattle 2018)

When Security takes over….



controlplane

When Security takes over….



controlplane

Determining Control Sets

Start simple!

More complex control sets require further:

- Automation
- Testing

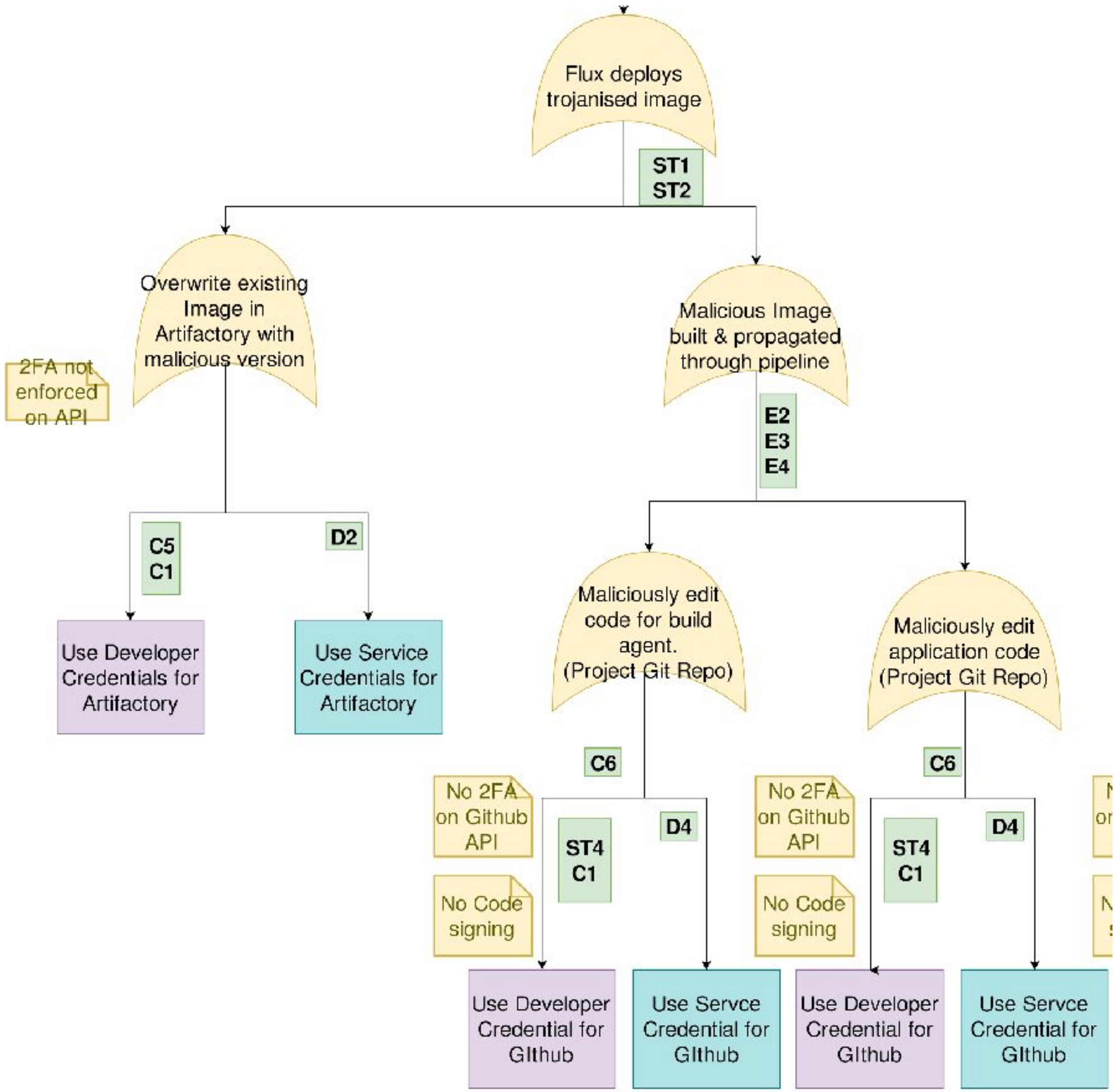
Risk is the determining factor

Determining Control Sets

Risk is the determining factor

Defence in depth with Attack Trees

KubeCon | CloudNativeCon Europe 2020 Virtual



Attack trees can demonstrate how seemingly unrelated controls can mitigate threats

ID	Control
ST1	in-toto Admission Controller
ST2	kubesec Admission Controller
ST4	gpg signed commits
C1	2FA
C5	Devs have read only container image registry access
C6	Protected Branches enforce Peer Review
D2	CI server has no overwrite permission in container image registry
D4	CI server has read only permission in Github
E2	Static Code analysis
E3	Image vulnerability scanning
E4	Dynamic Security testing

controlplane

Automated Testing

The only way to validate control implementation is through automated tests

Test the threat to be mitigated, not the specifics of the mitigating control

Security tests for DoD under development

[Proposal] DoD Kubernetes/Container Security Proposal #391

Closed **timfong888** opened this issue on 3 Jun · 9 comments

timfong888 commented on 3 Jun • edited

Description:
To have a comprehensive and exhaustive list of "controls" for the Department of Defense (DoD) to secure Kubernetes end-to-end programmatically (meaning it can be inspected and verified with code; and ideally fixed/patched/configured with code)

Integrating Kubernetes with a global SOC

1. Threat Model
2. Reproduce the attacks against test clusters repeatedly (Tests)
3. Gather the signals generated
4. Work with SOC to configure their SIEM
5. Re-run the test cases
6. Make sure the SOC lights up

Addressing Compliance Culture shock

Precedents and other standards are always helpful

- CIS Benchmarks & associated tooling
- GKE PCI DSS OS

Map controls to required compliance standards & policies

- Automated tests demonstrate compliance in near real-time
- One Control = One Automated Test = One Compliance requirement fulfilled

May need a program to rewrite/modify policy for cloud native

- Opportunity to automate tests for existing questionnaires

Gotchas - Node Segregation

	Nodes	Pods
Authorization	Union of all the permissions of everything on the node	Only what is needed by containers in the pod
Network Access	Union of all network access required by the node	Can be restricted per-application with NetworkPolicy, Istio, etc.
Monitoring	Measurements are made from within the node	Measurements may be made from outside the pod
Resource Usage	Strong isolation, depending on underlying infrastructure	Some isolation through cgroups, subject to noisy neighbors

controlplane

Gotchas - Service Mesh & PSPs

Required pod capabilities

If pod security policies are enforced in your cluster and unless you use the Istio CNI Plugin, your pods must have the NET_ADMIN and NET_RAW capabilities allowed. The initialization containers of the Envoy proxies require these capabilities.

Istio without CNI Plugin

- init containers require NET_ADMIN & NET_RAW capabilities
- requires relaxation of Pod Security Policies

Solution is to implement custom Pod Security Policy with allowlist using OPA

* Attack doesn't work with service mesh

controlplane

Gotchas - On Prem mindset in cloud

Introducing Cloud Native and Kubernetes into a large regulated organisation requires as much of a cultural change as a technological change.

Byproducts of on-prem mindset

- Heavily manual change control
- Restrictive architectures
- Reliance on detective controls

TLDR

- Threat Model
- Draw Attack Trees
- Apply Controls
- Test!
- Integrate with SOC

We’re Hiring!

Just like everyone else ;)

KubeCon | CloudNativeCon

Europe 2020

Virtual

KEEP CLOUD NATIVE CONNECTED

HELM