Powering Flexible Payments in the Cloud with Kubernetes

PAYBASE_

whoami

Ana Calin

Systems Engineer @Paybase

Twitter: @AnaMariaCalin

Table of Contents

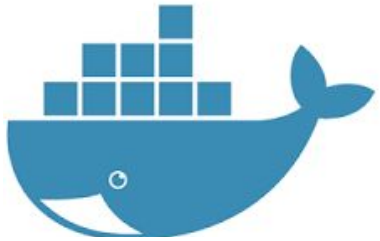
PAYBASE®_

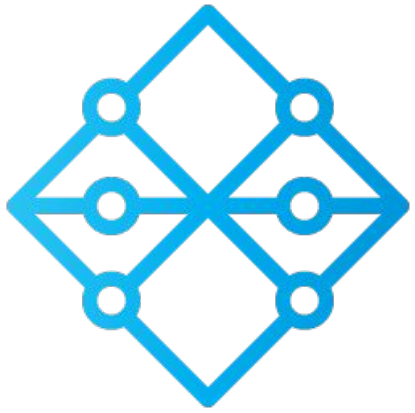
- API driven Payments Provider Platform
- B2B - marketplace, gig/sharing economies, cryptocurrency
- We make regulation easier for our customers

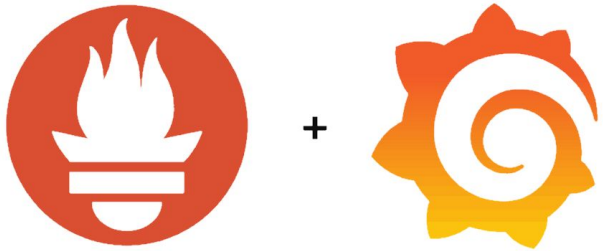
Things we've achieved so far

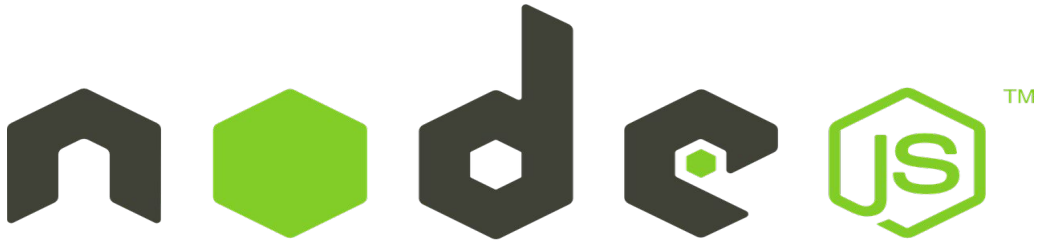
- ✓ We are ~ 2 years old
- ✓ Built our own processing platform from scratch
- ✓ We are currently onboarding our first 7 clients
- ✓ **FCA** authorised
- ✓ We have an **EMI** license
- ✓ Innovate UK grant worth £700k
- ✓ **PCI DSS** (The Payment Card Industry Data Security Standard) Level 1 compliant

Some of our tech stack









Anatomy of a compromise

Details about the compromise

- ✓ in the scope of an internal infrastructure penetration test
- ✓ in our production cluster
- ✓ pen tester had access to a privileged container

The weak link : GKE

- Compute engine scope
- Compute engine default service account
- Legacy metadata endpoints

- `oauth_scopes` - (Optional) The set of Google API scopes to be made available on all of the node VMs under the "default" service account. These can be either FQDNs, or scope aliases. The following scopes are necessary to ensure the correct functioning of the cluster:
  - `compute-rw` (`https://www.googleapis.com/auth/compute`)
  - `storage-ro` (`https://www.googleapis.com/auth/devstorage.read_only`)
  - `logging-write` (`https://www.googleapis.com/auth/logging.write`), if `logging_service` points to Google
  - `monitoring` (`https://www.googleapis.com/auth/monitoring`), if `monitoring_service` points to Google

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
automountServiceAccountToken: false
...
```

Metadata endpoints

```
~ kubectl exec -ti hopping-toad-fluentd-7f5fc7bc5-ll75w bash
root@hopping-toad-fluentd-7f5fc7bc5-ll75w:/# curl -s -H 'Metadata-Flavor: Google' 'http://
metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env' | grep ^KUBELET_
CERT | awk '{print $2}' | base64 -d
-----BEGIN CERTIFICATE-----
MIIC3DCCAcSgAwIBAgIRAPxCbwas4goGK6GKlrFK9w8wDQYJKoZIhvcNAQELBQAw
```

Mitigations

```
gcloud container clusters create [CLUSTER_NAME] \
  --service-account=$NODE_SA_EMAIL \
  --metadata disable-legacy-endpoints=true
```

OR

```
workload_metadata_config {
  node_metadata = "SECURE"
}
```

Result

```
~ kubectl exec -ti alternating-antelope-fluentd-6f45f6b67f-rcn52 bash
root@alternating-antelope-fluentd-6f45f6b67f-rcn52:/# curl -s -H 'Metadata-Flavor: Googl
e' 'http://metadata.google.internal/computeMetadata/v1/instance/attributes/kube-env'
This metadata endpoint is concealed.
root@alternating-antelope-fluentd-6f45f6b67f-rcn52:/#
```

The weak link : Tiller

- comes with mTLS disabled
- is able to create any K8S API resource in a cluster
- performs no authentication by default

Tiller

```
~ kubectl exec -ti hopping-toad-fluentd-7f5fc7bc5-ll75w bash
root@hopping-toad-fluentd-7f5fc7bc5-ll75w:/# helm version
Client: &version.Version{SemVer:"v2.13.0", GitCommit:"79d07943b03aea2b76c12644b4b54733bc5958d6", GitTreeState:"clean"}
Error: pods is forbidden: User "system:serviceaccount:default:default" cannot list pods in the namespace "kube-system"
root@hopping-toad-fluentd-7f5fc7bc5-ll75w:/# telnet tiller-deploy.kube-system 44134
Trying 10.28.5.108...
Connected to tiller-deploy.kube-system.svc.cluster.local.
Escape character is '^]'.
```

Mitigations

```
~ helm init \
    --upgrade \
    --service-account tiller \
    --override 'spec.template.spec.containers[0].command'='{/tiller,--storage=secret,--listen=localhost:44134}' \
    --wait
$HELM_HOME has been configured at /Users/anacalin/.helm.

Tiller (the Helm server-side component) has been upgraded to the current version.
Happy Helming!
```

RESULTS IN

```
~ kubectl exec -ti hopping-toad-fluentd-7f5fc7bc5-ll75w bash
root@hopping-toad-fluentd-7f5fc7bc5-ll75w:/# telnet tiller-deploy.kube-system 44134
Trying 10.28.5.108...
telnet: Unable to connect to remote host: Connection refused
root@hopping-toad-fluentd-7f5fc7bc5-ll75w:/#
```

Security and resilience

A secure K8S cluster should

- use a dedicated SA with minimal permissions
- use minimal scopes - least privilege principle
- use Network Policies or Istio with authorization rules set up
- use Pod Security Policies
- use scanned images
- have RBAC enabled

A resilient Kubernetes cluster should

- be architected with failure and elasticity in mind by default
- have a stable observability stack
- be tested with a tool such as Chaos Engineering

```
~ helm install --name chaoskube stable/chaoskube --set dryRun=false --set rbac.create
=true --set interval=15m
NAME:   chaoskube
LAST DEPLOYED: Sun Mar  3 08:14:06 2019
NAMESPACE: default
STATUS: DEPLOYED
```

Challenges we've encountered on our road to compliance

Challenge 1: The What

As **a PCI compliant PSP with many types of dbs**, I am want to be able to **query data-sets in a secure and db agnostic manner** so that **engineers and customers can use it easily** and **we are not prone to injections.(req. 6.5.1)**

Challenge 1: The How

Meet PQL

01 Inspired by SQL
02 Injection resistant
03 Used for querying data-sets
04 Database agnostic
05 Adheres to logical operator precedence

Challenge 1: The How

```
WHERE firstName LIKE %Rob% AND lastName NOT LIKE %Mugabe%
AND (isVIP = true OR occupation IN ("Extreme Unicyclist", "Space Lawyer"))
AND "Money Launderer" NOT IN tags AND age > 21
ORDER BY firstName ASC
```

01 Lexical analysis (tokenize input)

02 Syntactical analysis (parse tokenized input to AST)

03 Abstract Syntax Tree to specific database query

Challenge 2: The What

As **a PCI compliant PSP**, I am required to **implement only one primary function per server** to **prevent functions that require different security levels from coexisting on the same server.(req. 2.2.1)**

Challenge 2: The How

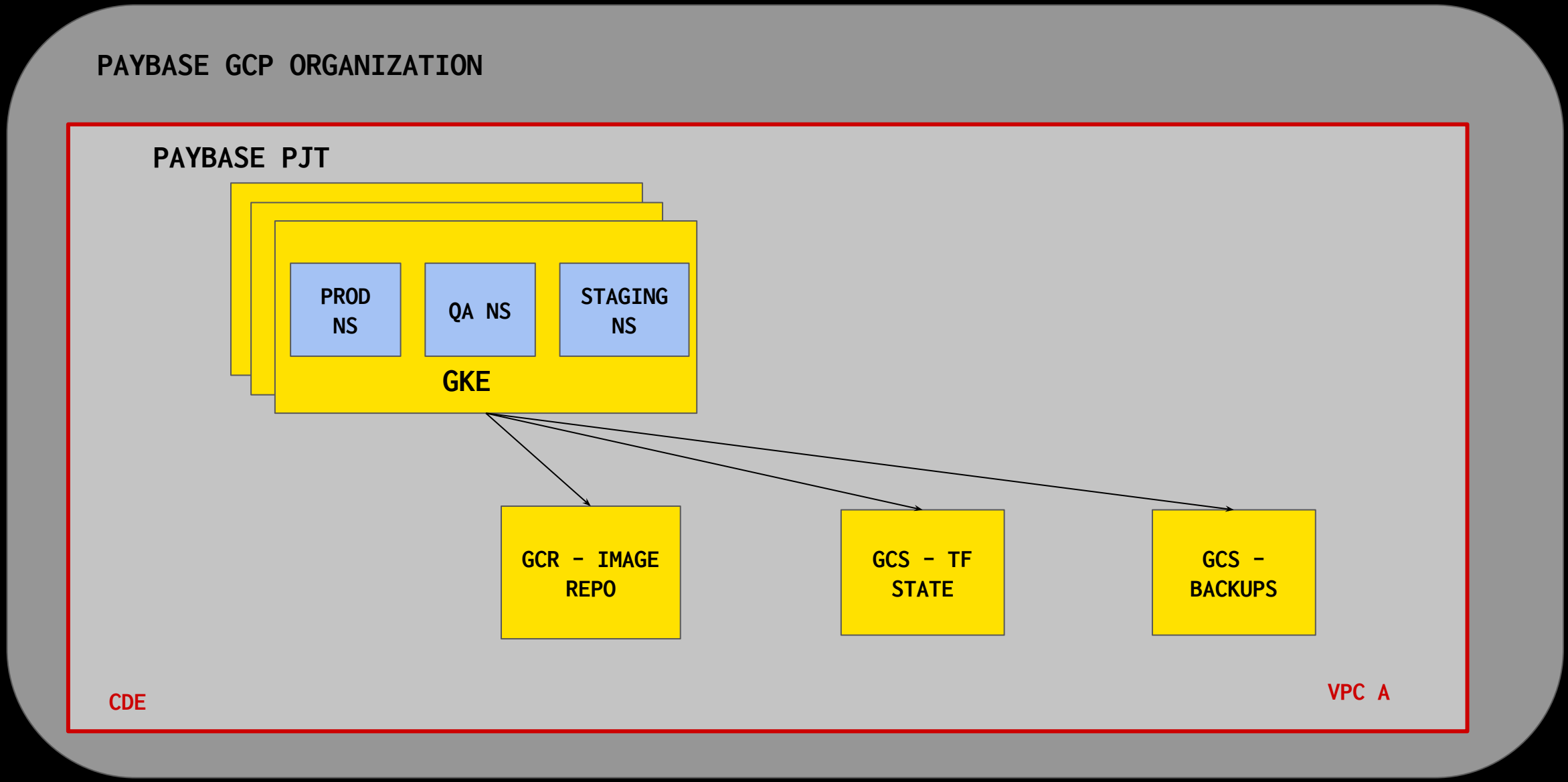
01 Server = Deployable Unit
02 Network Policies
03 Pod Security Policies
04 Only using trusted and approved images

Challenges we've circumvented on our road to compliance

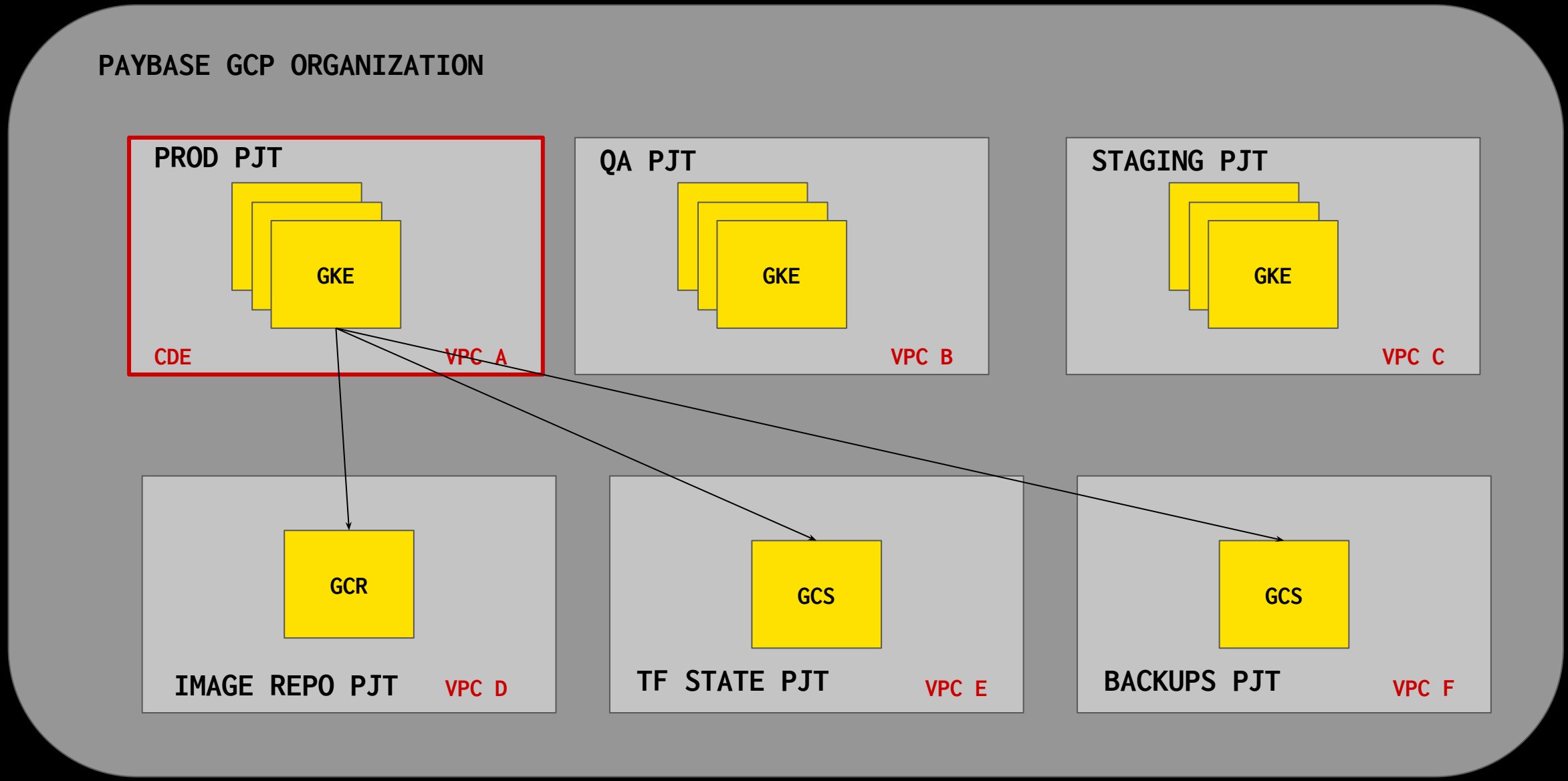
Challenge 3: The What

As **a PCI compliant PSP,** I am required **to remove all test data and accounts from system components before the system becomes active/goes into production (req.6.4.4)**

Common way of splitting environments



Paybase's way of splitting environments



Challenge 3: Benefit

01 Security

02 Separation of concerns

03 Reduction of PCI DSS scope

04 Easier to organize RBAC

Challenge 3: The What

As **a PCI compliant PSP,** I am required **to remove all test data and accounts from system components before the system becomes active/goes into production (req.6.4.4)**

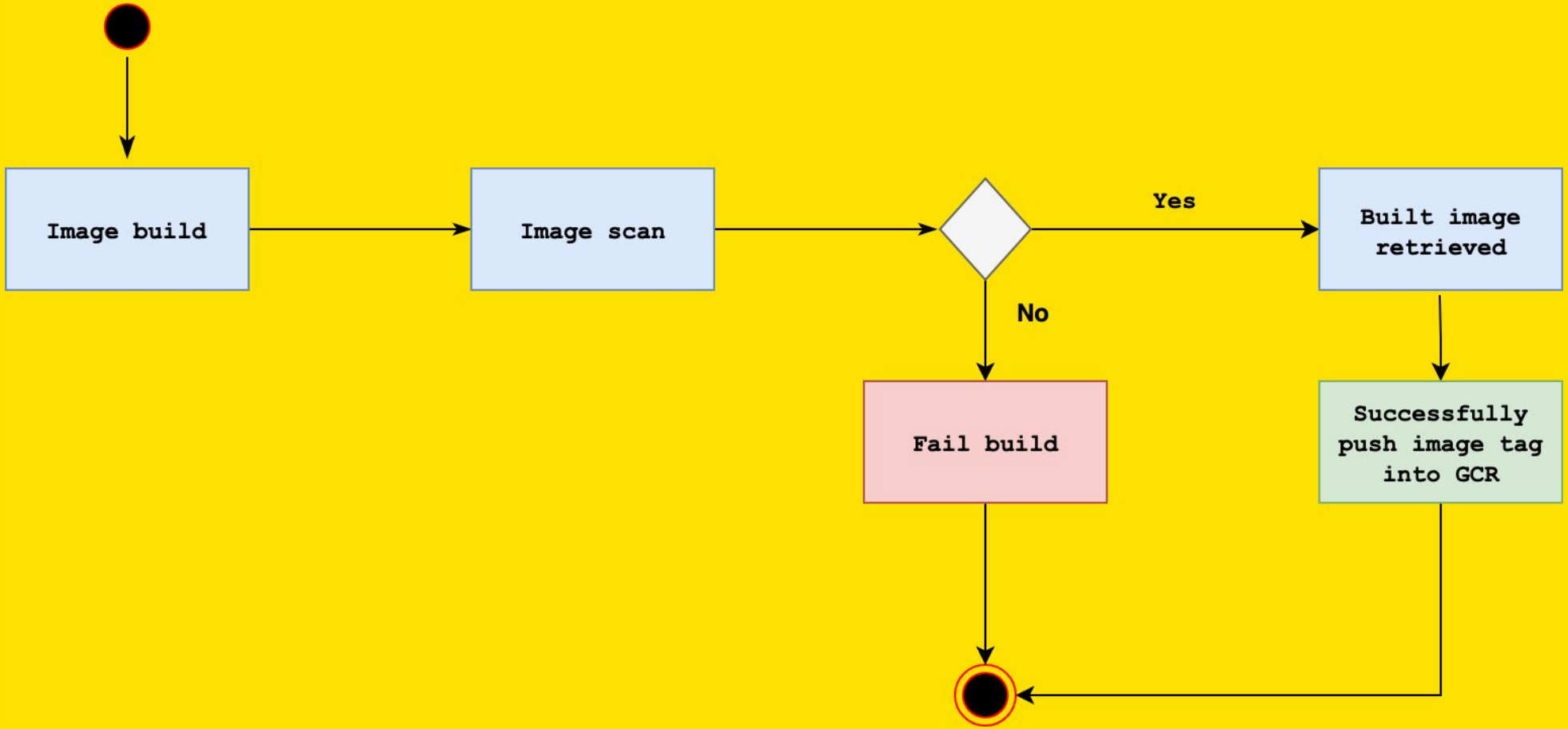
Challenge 4: The What

As **a PCI compliant PSP,** I am required **to perform quarterly internal vulnerability scans,address vulnerabilities and perform rescans to verify all "high risk" vulnerabilities are resolved in accordance with the entity's vulnerability ranking.(req.11.2.1)**

Challenge 4: The How

Image scanning

Here's a diagram



Summary

- security is not a point in time but an ongoing journey
- you can use OSS and achieve a good level of security
- we need to challenge the PCI DSS status quo

Resources

- ✓ https://www.4armed.com/blog/hacking-kubelet-on-gke/
- ✓ https://www.4armed.com/blog/kubeletmein-kubelet-hacking-tool/
- ✓ https://itnext.io/how-a-naughty-docker-image-on-aks-could-give-an-attacker-access-to-your-azure-subscription-6d05b92bf811

Thank you

<call to action here>

PAYBASE_