Breaking Kubernetes clusters

Martin Vladev - SAP

kubernetes

Gardener project

Provide Kubernetes Clusters-as-a-Service
homogeneously on hyper-scalers and on-premise
fully managed and with minimal TCO.

https://gardener.cloud

https://github.com/gardener/gardener

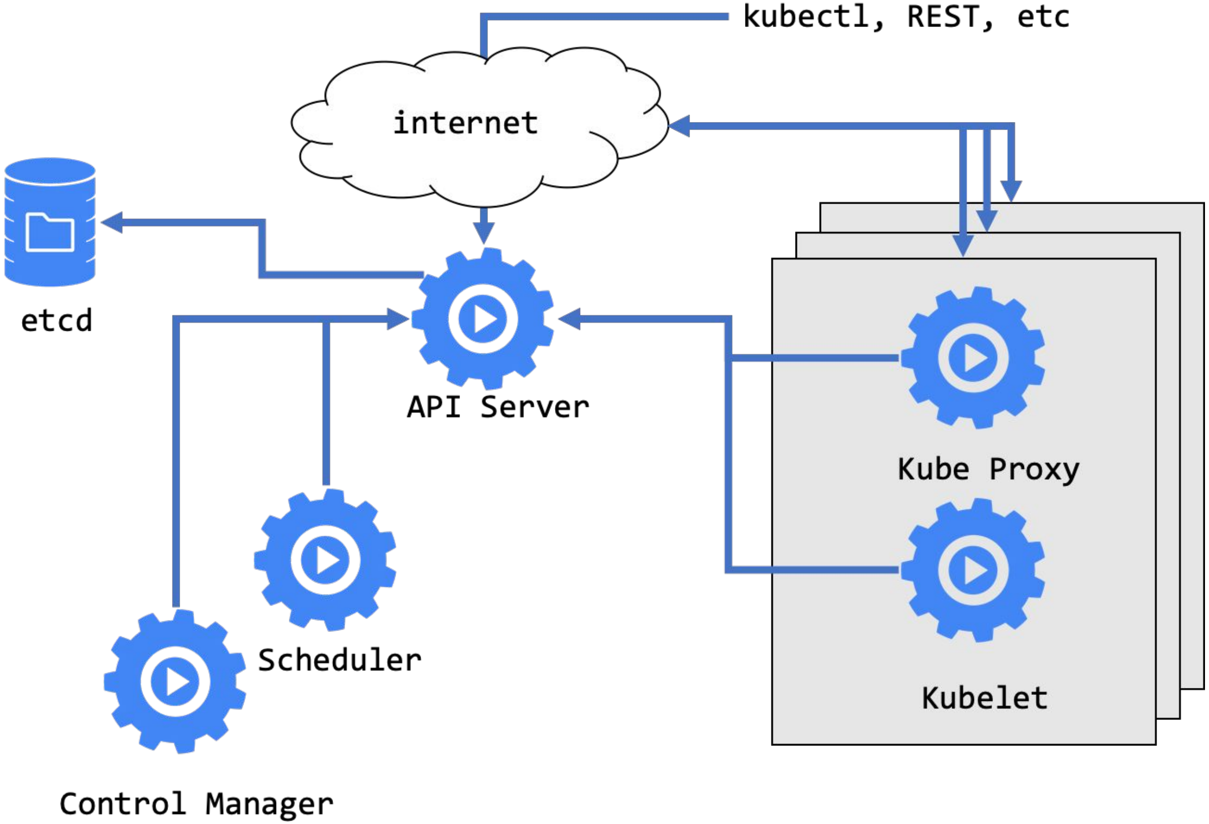
```
kubectl apply -f nuke.yaml
```

Personas

- Cluster Operator (**CO**)
  - manages the cluster (control plane and Nodes)
- Cluster Admin (**CA**)
  - assigns RBAC permissions to developers and creates cluster-wide resources
- Developer (**DEV**)
  - deploys workloads to specific namespaces
  - limited privileges

A person can have multiple personas depending on the organization / setup

K8S Architecture



Crash #1

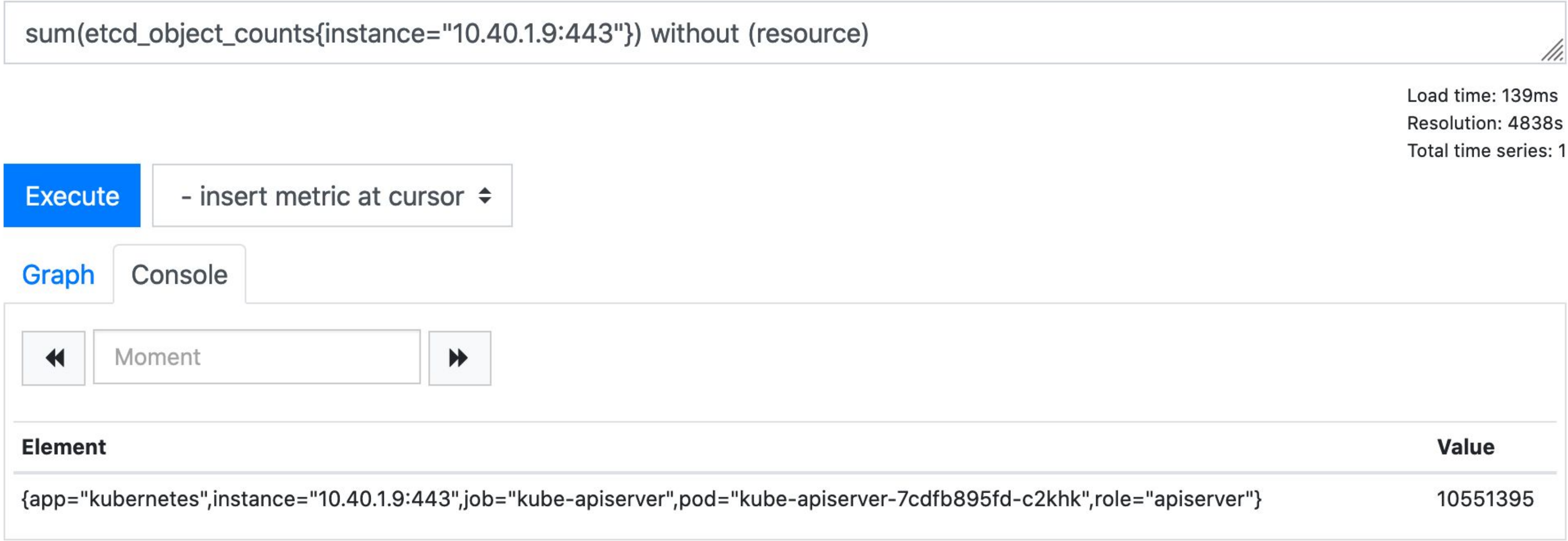
- **CA** integrates a CI/CD tool with the cluster.

- The CI/CD tool runs uses Helm(v2) on every build to deploy and upgrade their solution composed from many Helm Releases.

- After running for X amount of time, the API server stops responding.

Crash #1

- **CO** investigates and finds that **ETCD** is crashing, and looks at Prometheus for further information



```
sum(etcd_object_counts{instance="10.40.1.9:443"}) without (resource)
```

Load time: 139ms
Resolution: 4838s
Total time series: 1

Element	Value
{app="kubernetes",instance="10.40.1.9:443",job="kube-apiserver",pod="kube-apiserver-7cdfb895fd-c2khk",role="apiserver"}	10551395

Crash #1

Crash #1

CO resizes the ETCD cluster and deletes extra ConfigMaps

CA prevents this problem by adding limit to number of ConfigMaps in a namespace and upgrades to Helm v3

Tip:

```
count/*: "150"
```

Can be used for all objects

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: quota-configmaps
spec:
 hard:
   count/configmaps: "100"
```

Crash #2 - fix

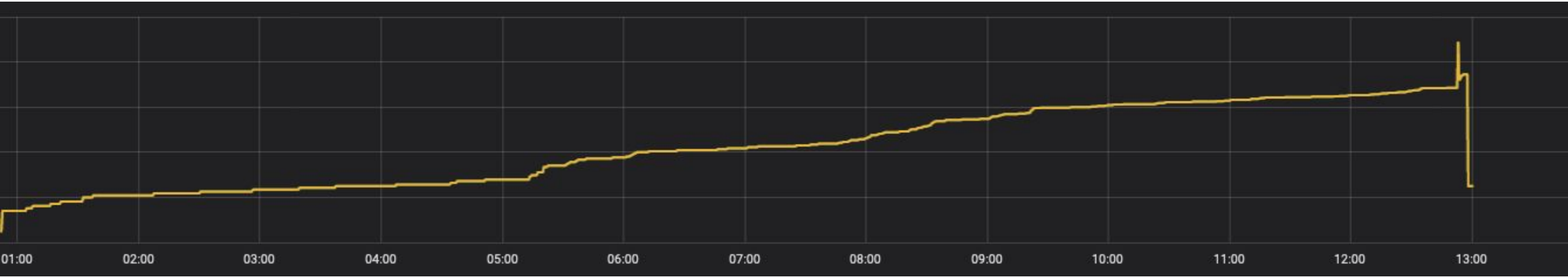
- **DEV** wants to store some application secret data in a cluster with 4 Nodes.

- The application uses Kubernetes Secret resource for that.

- After running for X amount of time, **kube-apiserver** stops responding.

Crash #2

- **CO** investigates and founds that **kube-apiserver** is getting OOM killed and evicted.
- And the application’s secret size (close to 1Mb each) is the problem.



Crash #2

```
   if s.Authentication.ServiceAccounts.Lookup {
       authenticatorConfig.ServiceAccountTokenGetter =
serviceaccountcontroller.NewGetterFromClient(
           extclient,
           versionedInformer.Core().V1().Secrets().Lister(),
           versionedInformer.Core().V1().ServiceAccounts().Lister(),
           versionedInformer.Core().V1().Pods().Lister(),
       )
   }
```

https://github.com/kubernetes/kubernetes/blob/v1.16.3/cmd/kube-apiserver/app/server.go#L529-L536

Crash #2

CO resizes the **kube-apiserver** temporary to give time for mitigations

DEV moves the data to dedicated secret store (such as Vault)

Tip:

If you still need to store such data in Kubernetes, it’s better to use a dedicated **CustomResource** for it as it’s content won’t be cached in any of the core Kubernetes controllers / apiserver.

Crash #3

- **DEV** decides to do a performance test on the cluster to see, if the cluster can support the application workload.

- All worker **Nodes** go down in a couple of seconds.

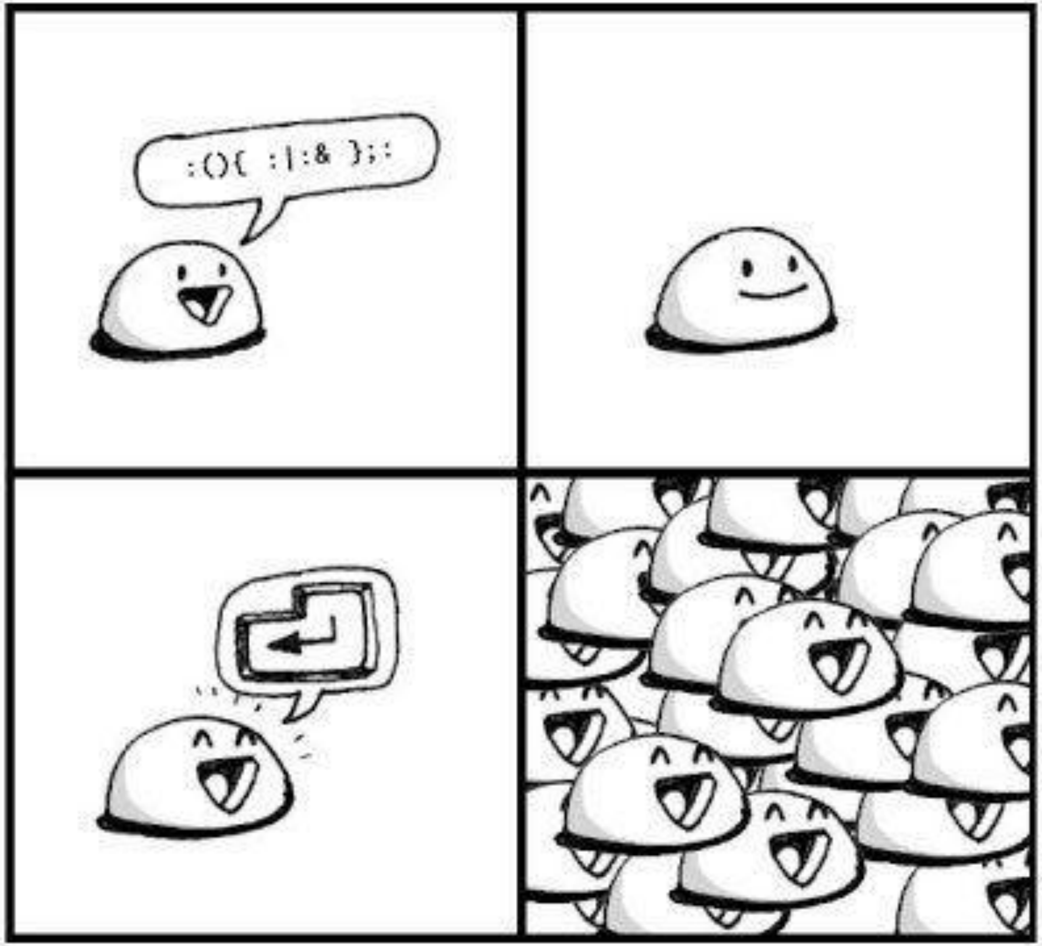
Crash #3

- It turned out that **DEV** ran the equivalent of executing the bash command bellow as a Pod in the cluster.

```
:(){ :|:& };:
```

The Pod was created by a DaemonSet...

Crash #3



Crash #3 - fix

DEV deletes the fork-bomb DaemonSet.

CO restarts all Nodes to bring them back from the grave.

Tip:

Set “**--pod-max-pids int**” and “**--feature-gates=SupportPodPidsLimit=true**” for automatic mitigation

Crash #4

- **DEV** adds a custom controller to the cluster.

- API server starts returning lots of 429 “Too many requests”

- Various components using the API cannot work - **kube-controller-manager, kube-scheduler, kubelet** ...

Crash #4

The controller had a bug where the QPS was set to too high - 3000 and it was spamming the API server

```
// QPS indicates the maximum QPS to the master from this client.
// If it's zero, the created RESTClient will use DefaultQPS: 5
QPS float32

// Maximum burst for throttle.
// If it's zero, the created RESTClient will use DefaultBurst: 10.
Burst int
```

https://github.com/kubernetes/client-go/blob/v12.0.0/rest/config.go#L110-L116

Crash #4 - fix

DEV updates the controller with fix

Tip:

Set “**--max-mutating-requests-inflight int int**” and “**--max-requests-inflight int**” in **kube-apiserver**

https://git.k8s.io/enhancements/keps/sig-api-machinery/20190228-priority-and-fairness.md

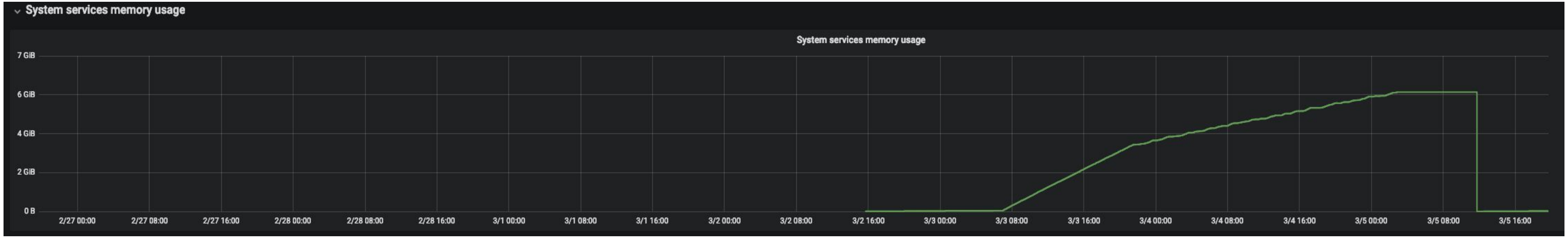
Crash #5

- **CA** starts rolling out **Nodes** with a new Operating System version to fix a CVE in **systemd-journald**

- After several days all **Nodes** start dying and have constant memory pressure

Crash #5

It turns that the CVE fix for **systemd-journald** caused a regression + memory leak:



https://github.com/systemd/systemd/issues/11900

Crash #5 - fix

CO rollbacks the change and waits for fix in upstream

Tip:

Wait for a little longer with dev / canary / prod deployment

Crash #6

- **CA** wants to use a service mesh to add more features to their solution.

- Istio is chosen and the sidecar injection is enabled by default - Envoy proxy is added to the workload automatically.

- When new **Nodes** are added to the cluster or existing components in **kube-system** namespace are changed, they stop working and the cluster slowly dies.

Crash #6

The problem was that the Envoy sidecar was added to ALL Pods in ALL namespaces which broke all system components - **kube-proxy**, **CNI**, **DNS**...

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
metadata:
 name: istio-sidecar-injector
webhooks:
- name: sidecar-injector.istio.io
 clientConfig:
   service:
     name: istio-sidecar-injector
     namespace: istio-system
     path: "/inject"
 rules:
 - {operations:["CREATE"],apiGroups:[""],apiVersions:["v1"],resources:["pods"]}
 failurePolicy: Fail
```

Crash #6

Fixes needed

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
metadata:
 name: istio-sidecar-injector
...
 failurePolicy: Fail
 namespaceSelector:
   matchLabels:
     Istio-injection: enabled
 timeoutSeconds: 5
```

Crash #6 - fix

CA had to fix the WebHook and delete all pods in **kube-system** namespace to restore them to the correct state

Tip:

Always make sure to not modify content in **kube-system** and the namespace in which the webhook is deployed.

Crash #7

- **CA** wants to improve the security of the cluster and make sure that no **Endpoint** points to an IP outside of the Pod CIDR.

- **DEV** writes a validating webhook which validates all **Endpoints** and deploys it in the cluster

- After some time **Deployments**, **StatefulSets** and other controllers stop working.

Crash #7

CO checks **kube-controller-manager** and sees

```
leaderelection.go:235] attempting to acquire leader lease  kube-system/kube-controller-manager…
```

On most clusters there is an **Endpoints** called **kube-system/kube-controller-manager** and it’s used for storing leader information.

Unfortunately the mutating webhook was evicted at some point and this led to the **kube-controller-manager** to lose leadership because it could not update it.

And finally - nothing can create the webhook Pod, because **kube-controller-manager** is not working.

Crash #7

Crash #7 - fix

CA labels **kube-system** namespace with **name=kube-system** and updates the **ValidatingWebhookConfiguration** to ignore **Endpoints** in it

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
...
 namespaceSelector:
   matchExpressions:
   - { key: name, operator: NotIn, values: [“kube-system”]}
```

Tip: always make sure that your webhook is running with multiple replicas

Crash #8

- **CA** wants to improve the security of the cluster and enable **NetworkPolicies** cluster-wide
- **CA** blocks all network traffic in namespaces and slowly enables traffic for each components
- DNS stops working.

Crash #8

CO investigates and discovers that **CA** has blocked **CoreDNS** from reaching the **kube-apiserver**.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: deny-all
 namespace: kube-system
spec:
 podSelector: {}
 policyTypes:
 - Egress
 - Ingress
 egress: []
 ingress: []
```

Crash #8 - fix

CA adds **NetworkPolicies** for all components which are managed

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: allow-to-apiserver
 namespace: kube-system
spec:
 podSelector:
   matchLabels:
     k8s-app: kube-dns
 policyTypes:
 - Egress
 egress:
 - to:
   - ipBlock: { cidr: 1.2.3.4/24 }
```

Tip:

Always delete existing **Pods** after applying **NetworkPolicies**.

Q&A

Thank you