AUSTRALIA
KUBERNETES COMMUNITY DAY
AUGUST 22

Managing Kubernetes Clusters on Spot Instances

Olga Mirensky, Platform Engineer, ANZx

Agenda

- Motivation for Spot Instances
  - Reserved Instances (RI), Committed User Discounts (CUD), etc.
  - Spot Instances
  - Focus on underlying VMs cost, not workload right-sizing
- Managing k8s and applications running on Spot
- Unknown unknowns
- Takeaways

Compute Backed by Availability SLA

- On-demand
- **Resource** Based
  - Reserved Instances (RI), Convertible, Committed Use Discounts (CUD)
  - AWS: 1yr - 40%, 3yr - **60%**
  - GCP: 1yr - 37%, 3yr - up to **57% / 70%**
- **Spend** Based
  - AWS Saving Plans: 3yr - up to **72%**
  - GCP Flex CUD: 1yr - 28%, 3yr - **46%**

Long-term commitment & doesn't cover 100% of your compute

Spot Instances 10,000 Foot Overview

	Google Cloud	aws	Azure
Discount	60 - 91%	70 - 90%	Up to 90%
Notice	30 sec	2 min, Rebalance recommendation	30 sec
Price Update	Once a month	Can be frequent	Variable
Options (not exhaustive)	One size fits all	Price and/or capacity optimised	Set your own max price
Price Insights	Difficult. API, Cost table	Easy. `aws ec2 describe-spot-price-history` Spot instance advisor	Portal price/eviction history, API

Common Patterns

Handling Spot Preemptions

✅ Stateless
✅ Interactive web applications
✅ Batch processing jobs

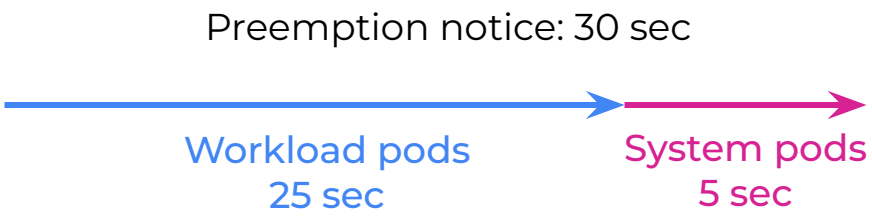
⚠️ Stateful
⚠️ CICD (e.g. terraform apply)

Application Graceful Shutdown

- SIGTERM handler
- Stop accepting new work
- Finalise in-flight work

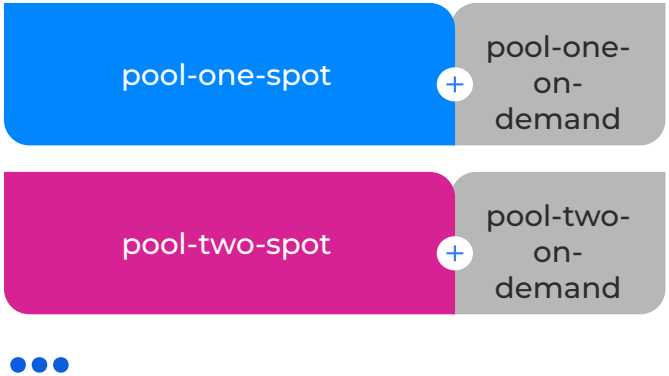
K8s Node Graceful Shutdown

- Node NotReady
- SIGTERM propagation



Spot Capacity Management in k8s

k8s nodepools:



Automatically labelled:

karpenter.sh/capacity-type: spot

eks.amazonaws.com/capacityType: SPOT

cloud.google.com/gke-spot: true

```yaml
nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: nodepool-name
        operator: In
        values:
        - pool-one-spot
        - pool-one-on-demand
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    preference:
      matchExpressions:
      - key: cloud.google.com/gke-spot
        operator: In
        values:
        - true
```

Capacity Management in k8s (cont)

Requirements

Pending pods: nodepools, affinities, etc

Constraints

available capacity

```yaml
# https://github.com/kubernetes/autoscaler
# higher number - higher priority (not %)
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*on-demand.*
    50:
      - .*spot.*
```

Cluster Autoscaler Priority-based expander,

GKE Cluster Autoscaler is price-optimised,

Karpenter,

Managed Dataplane,

...

Unexpected Twists

We Broke Everything (but not really)

	PF	READY	RESTARTS	STATUS	CPU	MEM	%CPU/R	%CPU/L	%MEM/R	%MEM/L
-66b95c8b9c-292qt	●	2/2	0	Running	5	243				
6754f-mhvd2	●	7/7	0	Running	22	730				
5w82	●	0/2	0	Completed	0	0				
t	●	0/0	0	NodeAffinity	0	0				
jlx	●	4/4	2	Running	1126	4056				
sjr	●	0/0	0	NodeAffinity	0	0				
7qn	●	0/0	0	NodeAffinity	0	0				
frm	●	4/4	0	Running	178	4206				
	●	0/2	0	Completed	0	0				
elop-865b95584d-lsn8p	●	2/2	0	Running	6	118				
-7d4df9dbf8-2q5dp	●	2/2	0	Running	5	240				
-7d4df9dbf8-95j8r	●	0/2	0	Error	0	0				
849696986-l4frx	●	2/2	0	Running	7	180				
6cpt	●	0/0	0	NodeAffinity	0	0	0	0	0	0
wf6x	●	4/4	2	Running	87	5196	1	1	30	28
pclg	●	0/0	0	NodeAffinity	0	0	0	0	0	0
psgk	●	4/4	2	Running	46	4449	1	0		
	●	0/2	0	Completed	0	0	0	0		
-jgflj	●	2/2	0	Running	6	124	1	0		
-xxw98	●	2/2	0	Running	6	117	1	0		
-zncds	●	2/2	0	Running	11	132	2	0		

Name	Status ↑	Type	Pods	Namespace	Cluster	Pods Running	Pods Desired
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2
	OutOfcpu	Deployment	2/2			2	2

ingressgateway-564594c67c-njcdm	0/1	NodeAffinity	0	6d15h
ingressgateway-564594c67c-nlrdp	1/1	Running	0	37h
ingressgateway-564594c67c-nz6j2	0/1	Completed	0	6d15h
ingressgateway-564594c67c-pqvdk	1/1	Terminated	0	6d15h
ingressgateway-564594c67c-psf8t	1/1	Running	0	2d9h
ingressgateway-564594c67c-q947l	1/1	Running	0	2d9h
ingressgateway-564594c67c-qcw2g	1/1	Running	0	6d15h
ingressgateway-564594c67c-qp8lj	0/1	NodeAffinity	0	6d15h
ingressgateway-564594c67c-qqg9n	0/1	NodeAffinity	0	6d15h
ingressgateway-564594c67c-t9fhm	0/1	NodeAffinity	0	6d15h
ingressgateway-564594c67c-v8wq8	1/1	Running	0	6d15h
ingressgateway-564594c67c-xf7dd	1/1	Running	0	2d9h
ingressgateway-564594c67c-xfbds	0/1	NodeAffinity	0	6d15h

OutOfpods, Error, NotReady, ContainerStatusUknown, NodeShutdown, Terminated, Init:ContainerStatusUnknown and more!! 🥱

NodeAffinity

```
$ kubectl get pod $name -o yaml
…
status:
  message: Pod Predicate NodeAffinity failed
  phase: Failed
  reason: NodeAffinity
```

```
Warning  NodeAffinity 41m                      kubelet  Predicate NodeAffinity failed
Warning  FailedMount  2m21s (x3870 over 5d10h) kubelet  MountVolume.SetUp failed for volume
                                        "xxxx": object "<namespace>"/"<name>" not registered
Warning  FailedMount  26s (x28 over 41m)       kubelet  MountVolume.SetUp failed for volume
                       kube-api-access-12345: object "my-ns"/"kube-root-ca.crt" not registered
```

Additional Challenges

- Preemption and availability is only the beginning.
  - Preempting many busy nodes at once causes many pods to start at the same time one a new node
    - OutOfcpu
    - OutOfpods
  - Automatic VM reclamation causes VM replacement for the same node and kubelet restart
    - NodeAffinity

How’s This Deployment Doing?

```
$ kubectl get pod -l k8s-app=my-app
NAME                        READY   STATUS          RESTARTS   AGE
my-app-564594c67c-njcdm     0/1     NodeAffinity    0          6d15h
my-app-564594c67c-nlrdp     1/1     Running         0          37h
my-app-564594c67c-nz6j2     0/1     Completed       0          6d15h
my-app-564594c67c-pqvdk     1/1     Terminated      0          6d15h
my-app-564594c67c-psf8t     1/1     Running         0          2d9h
my-app-564594c67c-qcw2g     1/1     Running         0          6d15h
my-app-564594c67c-qp81j     0/1     NodeAffinity    0          6d15h
my-app-564594c67c-qqg9n     0/1     NodeAffinity    0          6d15h
my-app-564594c67c-t9fhm     0/1     NodeAffinity    0          6d15h
my-app-564594c67c-v8wq8     1/1     Running         0          6d15h
```

Failed pods:
don’t consume resources, don’t count towards pods per node, don’t count in controllers.
Just objects in etcd.

```
$ kubectl get deploy -l k8s-app=my-app
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
my-app    4/4     4            4           48d
```

The only **real** harm these pods cause is confusion.

```
$ kubectl get pods --field-selector status.phase=Failed
```

Mitigations and Takeaways

Monitoring and Alerting

- Platform Critical User Journeys (CUJ) and SLOs
  - Confidence for platform consumers and Platform Engineering around platform stability.
  - Alert on error budget burn
- Monitoring
  - Rate of preemptions - insights and troubleshooting, but don’t alert.
  - Pod/node churn

Descheduler

https://github.com/kubernetes-sigs/descheduler

Finds pods that can be moved according to configurable policies and **evicts** them.

Example policies useful in Spot clusters:

- Remove Failed pods
- Rebalance Availability Zones
- Spread pods across nodes

Free Chaos Engineering

k8s best practices applicable to on-demand and crucial on Spot:

- Replication
- Spread across zones and nodes (TSC[1], pod AntiAffinity)
- Graceful shutdown
- Probes, especially StartUp
- Tier applications by priority
- PDBs. Don’t protect from Spot preemptions, but improve overall reliability

[1] new features: https://kubernetes.io/blog/2023/04/17/fine-grained-pod-topology-spread-features-beta/