The Many Ways of launching AWS spot instances

CKI edition

Michael Hofmann

overview

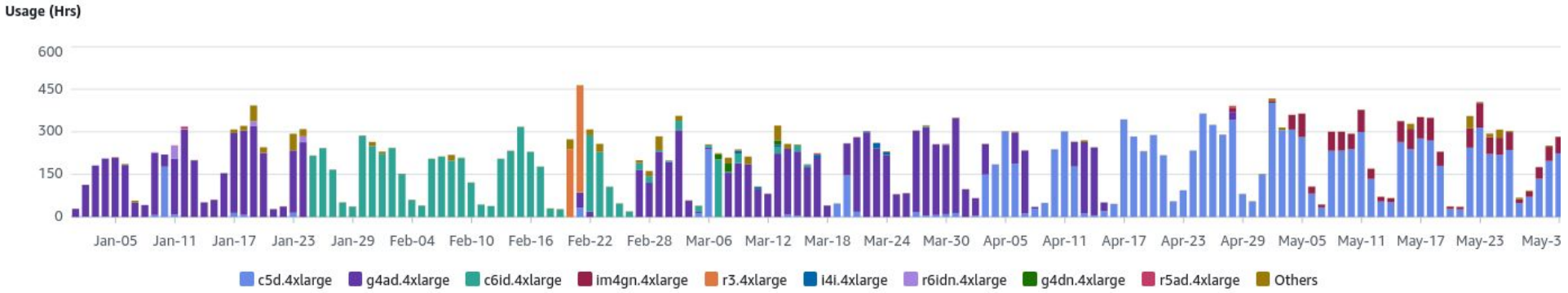
CKI against the failing spot instances (a fairy tale)

- why kernel CI pipelines use spot instances
- what it looks like when spot instances fail
- what are spot instances, actually
- a better way of requesting spot instances

introduction

introduction

- CKI: Continuous Kernel Integration - CI as a service
  - CI infrastructure for Red Hat's kernel development
  - prevent bugs from being merged into kernel trees
  - home page and documentation: https://cki-project.org
  - code: https://gitlab.com/cki-project
- GitLab CI pipeline: build (AWS spot instances), test (Beaker lab)
  - build x86_64/aarch64/ppc64le/s390x: ~300 hours/workday
  - native/cross-arch building: 16vCPU/32GiB/SSD

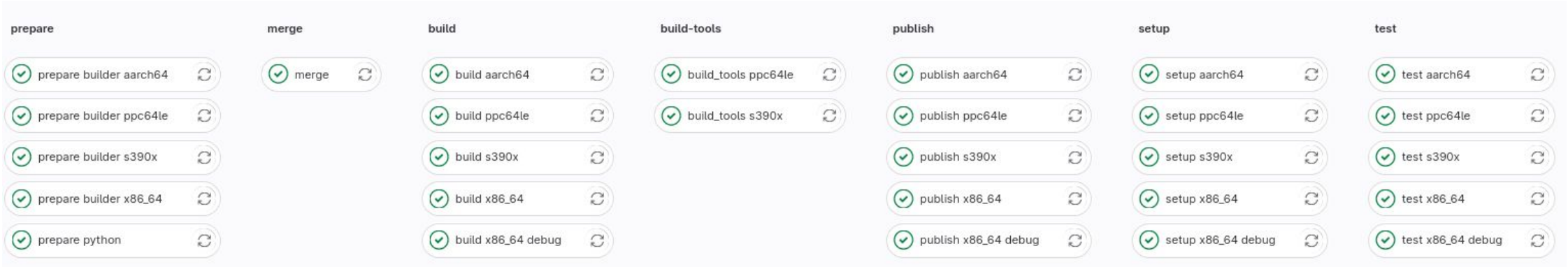


Red Hat

GitLab pipeline jobs on AWS spot instances

in a nutshell

- GitLab jobs -> gitlab-runner -> docker-machine -> EC2 instances
- docker-machine: create server, install Docker, forward socket
  - abandoned upstream, GitLab fork
  - ec2.RequestSpotInstances(InstanceType=c5d.4xlarge, az=a)



Red Hat

end of 2022: when (spot) instances fail...

random job failures

There has been a runner system failure, please try again

```
104 Authenticating with credentials from job payload (GitLab Registry)
105 Pulling docker image registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper:x86_64-c6bb62f6 ...
106 WARNING: Failed to pull image with policy "always": Cannot connect to the Docker daemon at tcp://10.29.162.188:2376. Is the docker daemon running? (manager.go:203:0s)
108 ERROR: Failed to cleanup volumes
109 ERROR: Job failed (system failure): Cannot connect to the Docker daemon at tcp://10.29.162.188:2376. Is the docker daemon running?
```

~30% of jobs failing...

```
2022-12-01 12:14:58 😈 P710288917 J3407759119 build aarch64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:18:34 😈 P710320034 J3407878231 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:20:10 😈 P710346311 J3407926081 build aarch64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:20:26 😈 P710344002 J3407926151 build aarch64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:20:28 😈 P710340177 J3407926123 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:20:56 😈 P710344002 J3407926167 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:21:59 😈 P710362391 J3407939603 build ppc64le failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:30 😈 P710351557 J3407940734 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:33 😈 P710356382 J3407940764 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:48 😈 P710362390 J3407945731 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:50 😈 P710346311 J3407945983 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:53 😈 P710356382 J3407945781 build ppc64le failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:55 😈 P710356382 J3407946214 build aarch64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:22:59 😈 P710362391 J3407946921 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:23:18 😈 P710356382 J3407947972 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:23:22 😈 P710351557 J3407948191 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:23:28 😈 P710362393 J3407948249 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:23:30 😈 P710362391 J3407948298 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:24:25 😈 P710362393 J3407956334 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:24:31 😈 P710367040 J3407955650 build aarch64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:24:53 😈 P710320034 J3407956964 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:24:55 😈 P710320034 J3407957542 build ppc64le failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:25:01 😈 P710320034 J3407957561 build aarch64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:31:07 😈 P710340177 J3407978412 build ppc64le failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:31:16 😈 P710344002 J3407980292 build s390x failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:31:18 😈 P710344002 J3407978906 build ppc64le failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:32:17 😈 P710344004 J3407983136 build x86_64 failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:32:19 😈 P710344004 J3407980419 build x86_64 debug failed: Detected System failure, retrying in 10 minutes
2022-12-01 12:32:22 😈 P710344004 J3407983291 build ppc64le failed: Detected System failure, retrying in 10 minutes
```

what happened?

unknown unknowns

- some digging later:

```
Error launching instance: UnfulfillableCapacity - There is no
capacity available that matches your request.
```

- switching to "normal" on-demand instances did not help:

```
Error launching instance: InsufficientInstanceCapacity - We
currently do not have sufficient c5ad.4xlarge capacity in the
Availability Zone you requested (us-east-1a). Our system will be
working on provisioning additional capacity.
```

- at least switching instance types helped for a short while 😰

so what are spot instances, actually

The Cloud a.k.a. Hyperscalers

- Wikipedia:
  - Hyperscale is the ability of an architecture to scale appropriately as increased demand is added to the system
- The Internet on the scale of the Amazon Cloud:
  - spans 99 Availability Zones within 31 geographic regions
  - ~ 125 physical data centers, >50k servers each

Amazon EC2 in a nutshell

- Elastic Compute Cloud (EC2): servers (instances) for rent
- from most expensive to cheapest (simplified):
  - On-demand Instances: pay as you go
  - Spot Instances: use spare EC2 capacity, can be interrupted
  - Compute Savings Plans: committed spend for compute
  - Reserved Instances: committed spend per instance family
- recommended strategy:
  - savings plans/reserved instances for steady workloads
  - spot instances for bursty interruptible workloads
  - on-demand instances for everything else

"interruptible"

there ain't no such thing as a free lunch

Region: US East (Ohio) | OS: Linux/UNIX

Instance type filter:

vCPU (min): 16 | Memory GiB (min): 32 | Instance types supported by EMR

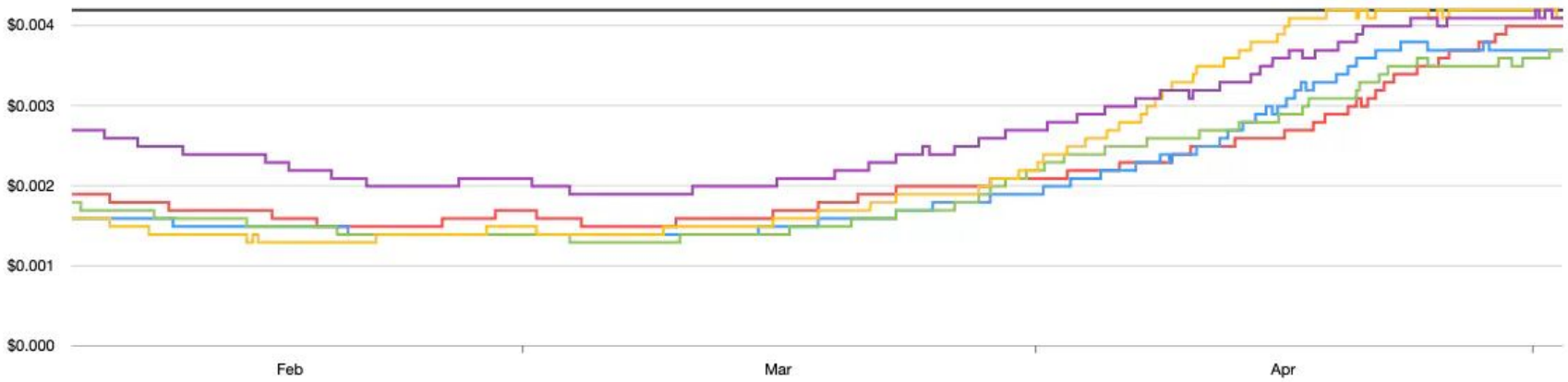
Instance Type	vCPU	Memory GiB	Savings over On-Demand*	Frequency of interruption
m5.16xlarge	64	256	58%	<5%
i3en.12xlarge	48	384	70%	<5%
m6gd.metal	64	256	73%	<5%
r5d.24xlarge	96	768	83%	<5%
r5d.12xlarge	48	384	74%	<5%
m6g.16xlarge	64	256	60%	<5%
m5.8xlarge	32	128	71%	<5%

- sometimes AWS wants its instances back
  - Cloud vs. Reality
- 2 minutes notice before termination
- chance of interruption varies by instance type
  - AWS spot instance advisor
- so < 5% doesn't sound to bad 🤔

demand for instance families

...all just my assumptions below...

- fixed number of racks per instance family per datacenter
- spare capacity per instance family
  - otherwise on-demand requests for specific instances types fail
  - spare capacity can be temporarily used by spot instances
  - spot instance users could be steered by price



Red Hat

intermediate conclusion

- UnfulfillableCapacity/InsufficientInstanceCapacity - not enough capacity in AWS data centers
- CKI RequestSpotInstances() requests were very specific:
  - ec2.RequestSpotInstances(InstanceType=c5d.4xlarge, az=a)
  - specific instance type, datacenter
  - cost, availability not taken into account
- so how to fix that?
  - flexible instance types, data centers
  - request cheap but also available instance types

a better way of requesting spot instances

API calls that can launch spot instances

hysterical raisins

- AWS documentation is pretty outspoken on what is recommended
- Amazon EC2 Spot Workshop shows how to try them out

API call	recommended by AWS	multiple instance types/datacenters	ease of use for CI workers
RunInstances	no	no	easy
RequestSpotInstances	no	no	medium
RequestSpotFleet	no	yes	medium
CreateFleet	yes	yes	medium
CreateAutoScalingGroup	yes	yes	hard

RequestSpotInstances vs. CreateFleet

- CreateFleet: mandatory launch templates

Capability	RequestSpotInstances	CreateFleet
Directly specify instance properties	yes	no
Launch template support	yes	yes
Launch template overrides	no	yes
Instance requirements	no	yes
Allocation strategies	no	yes

trying this out: create launch template

- launch template data YAML:

```
ImageId: ami-1234567890
KeyName: some-ec2-key
InstanceRequirements:
  VCpuCount: {Min: 16}
  MemoryMiB: {Min: 32768}
  ExcludedInstanceTypes: [m1.*]
  LocalStorageTypes: [ssd]
```

- call the API:

```
aws ec2 create-launch-template \
  --launch-template-name some-launch-template-name \
  --launch-template-data file://launch-template-data.json
```

trying this out: 11 matching instance types

Instance type	vCPUs	Memory (GiB)
c5d.4xlarge	16	32.00
c6id.4xlarge	16	32.00
g4ad.4xlarge	16	64.00
g4dn.4xlarge	16	64.00
g5.4xlarge	16	64.00
m5ad.4xlarge	16	64.00
m5d.4xlarge	16	64.00
m5dn.4xlarge	16	64.00
m6id.4xlarge	16	64.00

trying this out: create fleet with two subnets

- create fleet config YAML:

```
Type: instant
TargetCapacitySpecification:
  DefaultTargetCapacityType: spot
  TotalTargetCapacity: 1
LaunchTemplateConfigs:
  - LaunchTemplateSpecification:
      LaunchTemplateName: some-launch-template-name
      Version: '$Default'
    Overrides:
      - SubnetId: subnet-12345
      - SubnetId: subnet-23456
```

- call the API:

```
aws ec2 create-fleet \
  --cli-input-json file://fleet-config.json
```

details and summary

- keeping up to date with AWS API is hard, but worth it
- CreateFleet() to robustly spawn instant spot/on-demand instances
  - requires launch templates
- instance requirements might select "interesting" instances, e.g.
  - [mc]1.* instances are not compatible with RHEL 10
  - [mrc]3.* non-NVMe SSDs must be explicitly initialized
- default price-capacity-optimized allocation strategy
  - cheapest available spot instance across subnets/data centers
- not supported in a lot of places yet
  - available in the CKI fork of docker-machine: merge request


🤗 Question time 🤗