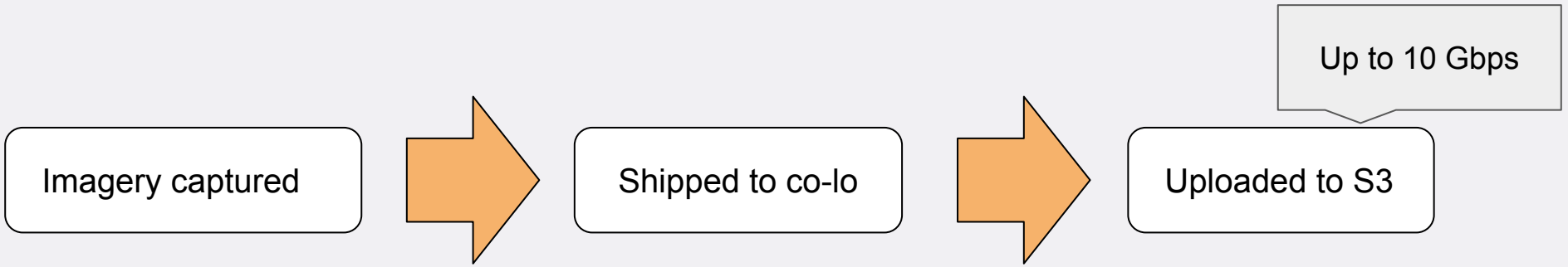
Using Terraform with AWS Batch for data processing at scale

Craig Loftus, Senior Developer

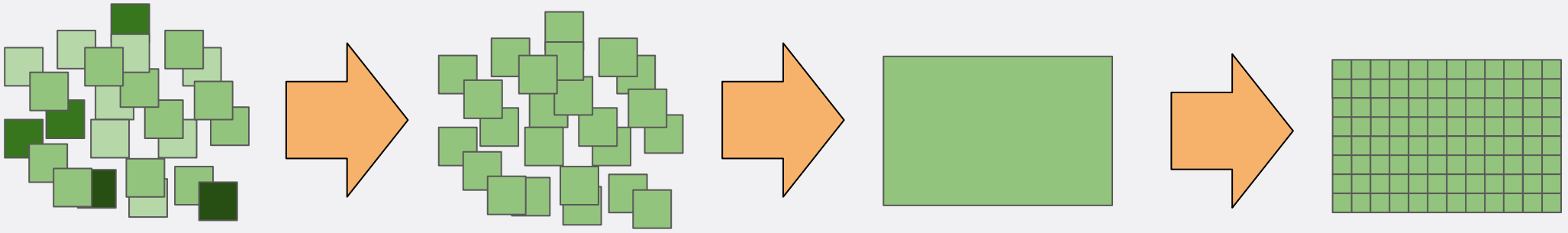
Terraform and AWS Batch

Data processing at scale

Data ingestion



Processing



- Different resource requirements
- Different scaling characteristics

Pipeline v1

- Use existing python scripts
- Refactor to accept S3 in and out urls
- Wrap each in docker
- Terraform to setup AWS Batch
- Python script to kick off AWS Batch jobs

Terraform - set up

- From scratch in Terraform
- Just a `main.tf` and a `variables.tf`
- About 300 lines of config
- So what does that actually look like?

```
provider "aws" {
  region = var.region
  allowed_account_ids = var.allowed_account_ids
}
```

```
terraform {
  backend "s3" {
    bucket = "bucket"
    key = "prod/pipeline.tfstate"
    dynamodb_table = "terraform-state-lock"
    encrypt = true
    region = "ap-southeast-2"
  }
}
```

```
resource "aws_ecr_repository" "pipeline" {
  name = "pipeline"
}
```

```
resource "aws_ecr_repository" "pipeline" {
  name = "pipeline"
}

output "repository_url" {
  description = "The URL for the pipeline ECR repository"
  value = aws_ecr_repository.pipeline.repository_url
}
```

```
data "aws_iam_policy_document" "pipeline_job_assume" {
  ...
}

resource "aws_iam_role" "pipeline_job" {
  assume_role_policy = data.aws_iam_policy_document.pipeline_job_assume.json
}
```

```
data "aws_iam_policy_document" "pipeline_job" {
  ...
}

resource "aws_iam_role_policy" "pipeline_job" {
  role = aws_iam_role.pipeline_job.name
  policy = data.aws_iam_policy_document.pipeline_job.json
}
```

```
resource "aws_iam_role_policy_attachment" "pipeline_job" {
  role = aws_iam_role.pipeline_job.name
  policy_arn = ".../service-role/AmazonEC2ContainerServiceforEC2Role"
}
```

```
resource "aws_iam_instance_profile" "pipeline_job" {
  role = aws_iam_role.pipeline_job.name
}
```

```
resource "aws_security_group" "pipeline" {
  name = "pipeline-security-group"

  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
data "aws_ssm_parameter" "ecs_amazon_linux_2" {
  name = "/aws/service/ecs/optimized-ami/amazon-linux-2/recommended"
}

resource "aws_launch_template" "pipeline" {
  name = "pipeline"
  ebs_optimized = true
  image_id = jsondecode(data.aws_ssm_parameter.ecs_amazon_linux_2.value).image_id
}
```

```
resource "aws_batch_compute_environment" "pipeline" {
  compute_environment_name_prefix = "pipeline"

  compute_resources {
    instance_role = aws_iam_instance_profile.pipeline_job.arn
    instance_type = ["p3.8xlarge", "m5.16xlarge",]

    max_vcpus = 96
    min_vcpus = 0

    security_group_ids = [aws_security_group.pipeline.id,]
    subnets = [var.subnet_id,]

    launch_template {
      launch_template_id = aws_launch_template.pipeline.id
      version = aws_launch_template.pipeline.latest_version
    }

    type = "EC2"
  }

  service_role = aws_iam_role.pipeline_service.arn
  type = "MANAGED"

  lifecycle {create_before_destroy = true}
  depends_on = [aws_iam_role_policy_attachment.pipeline_service]
}
```

```
resource "aws_batch_job_queue" "stitcher" {
  name = "stitcher"
  state = "ENABLED"
  priority = 1
  compute_environments = [aws_batch_compute_environment.pipeline.arn]
}
```

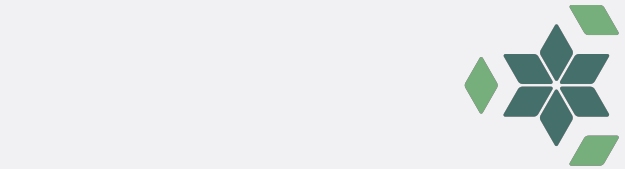
```
resource "aws_batch_job_definition" "stitcher" {
  name = "stitcher"
  type = "container"
  container_properties = jsonencode({
    image = "${aws_ecr_repository.pipeline.repository_url}:1.0"
    command = ["stitcher"]
    vcpus = 10
    memory = 50000
    resourceRequirements = [
      {
        type = "GPU"
        value = "2"
      },
    ]
  })
}
```

Make it go now

```
$ terraform apply
…
Outputs:

repository_url = <url>

$ docker tag <image_id> <url>:1.0
$ docker push <url>:1.0
$ aws batch submit-job --job-name=”Foo” --job-queue=”stitcher” --job-definition=”stitcher”
```



Lessons

- Get up and running with Terraform and Batch quickly
- AWS launch template not well supported
- External dependencies are the enemy
- Batch status monitoring and debugging is poor
- Batch limits interfaces between jobs to cli options and success or fail status.
- Batch limited to 20 dependant jobs

Thank you. Any questions?

craig@dendra.io
@craigloftus on DigitalOxford