re:Invent new demo

START WITH BROWSER OPEN TO VPC CONSOLE and have VS CODE editor open to the fugue-field-demo-kit project folder and CLI open.

All the code is here: https://github.com/fugue-sage/fugue-field-demo-kit

Before the demo, be sure "staging", "dev-hector", and "dev-lisa" processes are running. Otherwise run:

```
fugue run VPC-hector.lw -a dev-hector
fugue run Staging.lw -a staging
fugue run VPC-lisa.lw -a dev-lisa
```

Story Overview:

Sam is the new hire. As part of his onboarding, Fugue is used to automatically deploy a VPC for his personal AWS lab.

Lisa has been promoted and no longer needs her environment.

Hector is up and running and working on integrations.

The demo will start with the CLI, to show status and to kill

Lisa’s VPC and add Sam’s.

We’ll use the *history* command against Sam’s newly launched VPC as a way to introduce how the conductor enforcement is running as a system job.

Hector is working on integrations and having some issues communicating with a third-party API and adds a rule to open All Traffic from the world for his SG, which we’ll use as the example for enforcement. Add that rule, then switch to the composition to highlight the SG permission blocks, as an intro to the code as well as to burn a little talking time while waiting for the conductor tick.

Show the refresh of the SG and the rule being removed, then explore imports and compile-time validation.

Finally, we’ll update the SG for Sam with an SSH rule since he needs to work directly on some instances in his VPC.

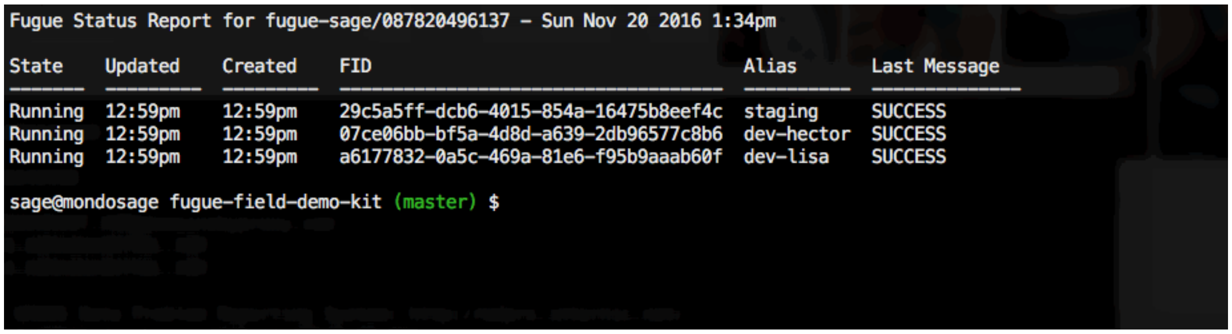
<u>Detailed Narrative</u>

Typed commands are listed with backticks.
Actions are in parentheses.
Screenshots are there to illustrate what should be shown at that point.

First, let’s check to see what’s running:

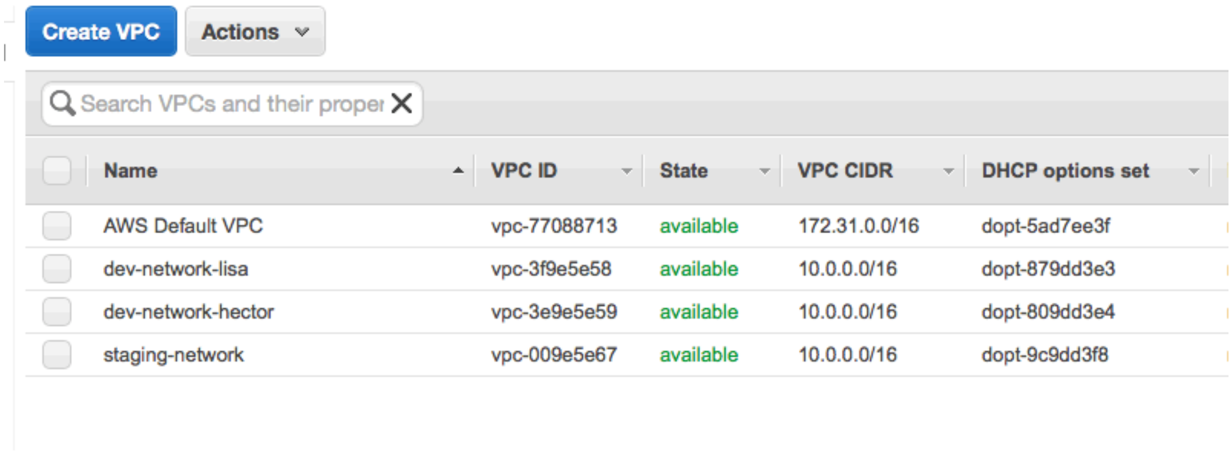
`fugue status`

Here we see three fugue processes. One for Lisa, one for Hector, and one for the staging environment.



```
Fugue Status Report for fugue-sage/087820496137 - Sun Nov 20 2016 1:34pm

State    Updated    Created    FID                                   Alias       Last Message
-------  ---------  ---------  ------------------------------------  ----------  ------------
Running  12:59pm    12:59pm    29c5a5ff-dcb6-4015-854a-16475b8eef4c  staging     SUCCESS
Running  12:59pm    12:59pm    07ce06bb-bf5a-4d8d-a639-2db96577c8b6  dev-hector  SUCCESS
Running  12:59pm    12:59pm    a6177832-0a5c-469a-81e6-f95b9aaab60f  dev-lisa    SUCCESS

sage@mondosage fugue-field-demo-kit (master) $
```

And here in the web console we see each of the VPCs.



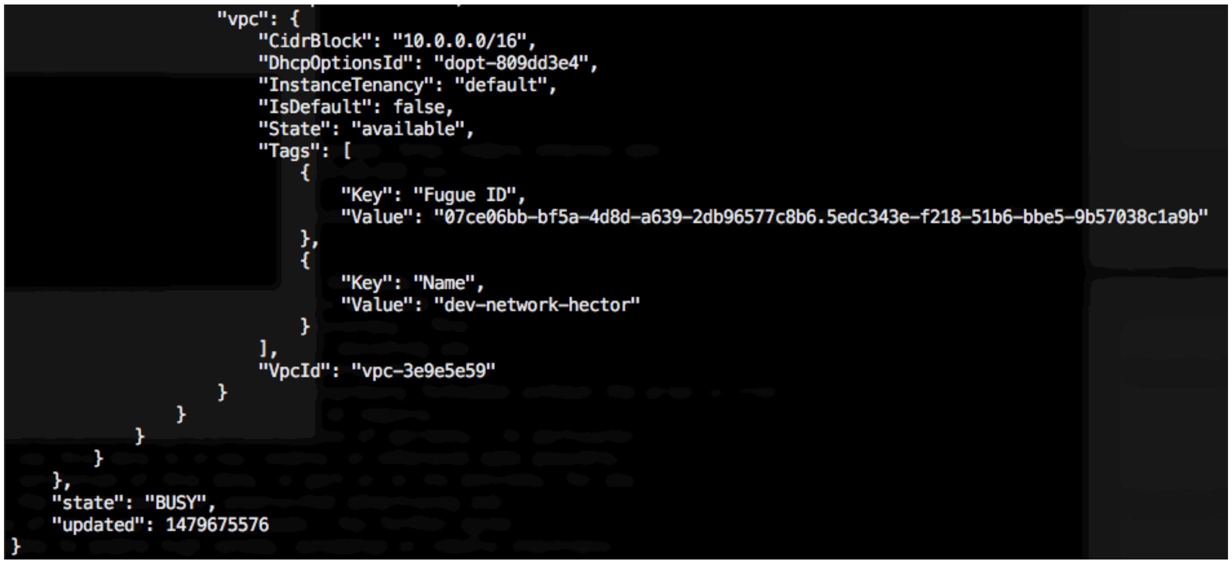
Name	VPC ID	State	VPC CIDR	DHCP options set
AWS Default VPC	vpc-77088713	available	172.31.0.0/16	dopt-5ad7ee3f
dev-network-lisa	vpc-3f9e5e58	available	10.0.0.0/16	dopt-879dd3e3
dev-network-hector	vpc-3e9e5e59	available	10.0.0.0/16	dopt-809dd3e4
staging-network	vpc-009e5e67	available	10.0.0.0/16	dopt-9c9dd3f8

(Back to the terminal)

Each cloud workload is run as a process by the Fugue Conductor. You can check each by using the *fugue status <process>* command, which outputs a list of everything running in that workload. Let’s see what that looks like for

Lisa’s VPC.

`fugue status dev-lisa`

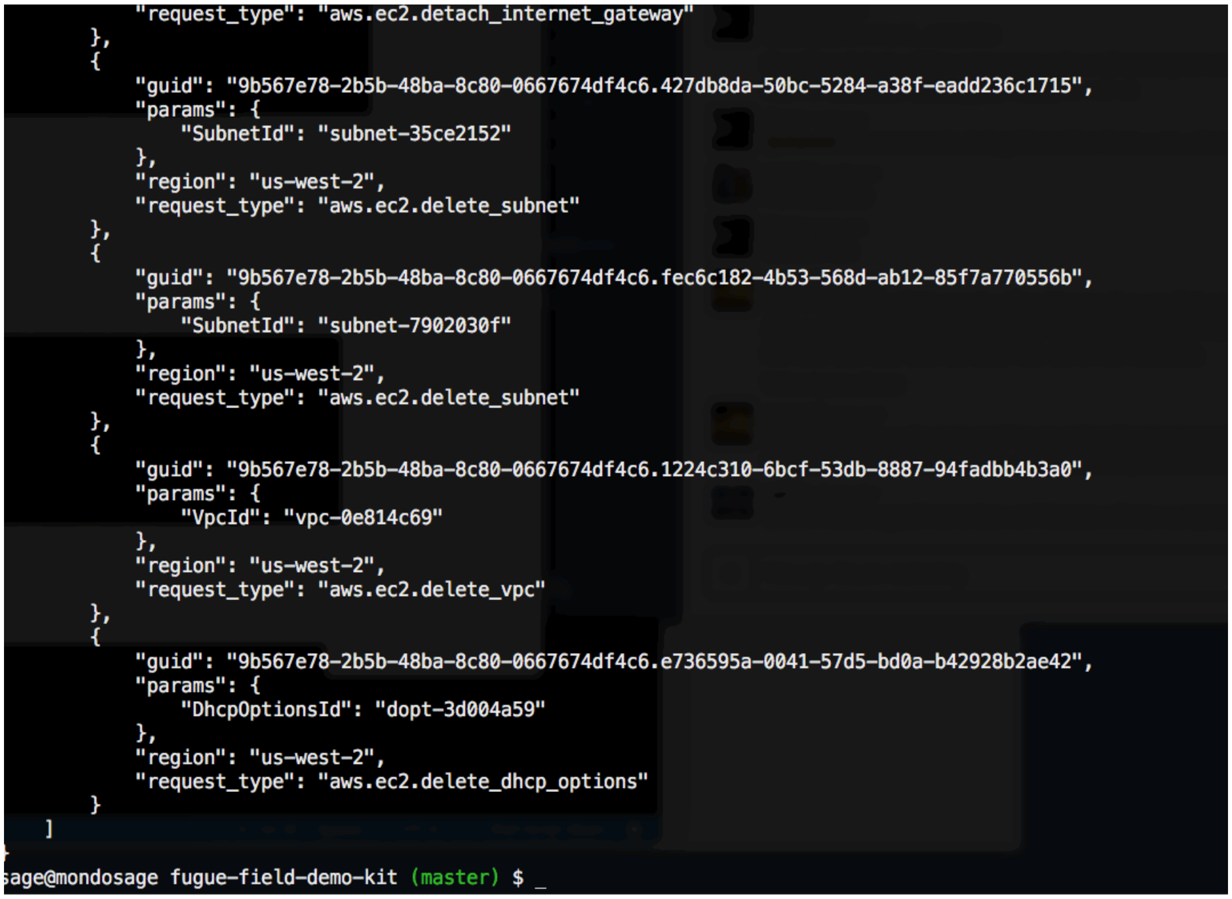


```
                "vpc": {
                    "CidrBlock": "10.0.0.0/16",
                    "DhcpOptionsId": "dopt-809dd3e4",
                    "InstanceTenancy": "default",
                    "IsDefault": false,
                    "State": "available",
                    "Tags": [
                        {
                            "Key": "Fugue ID",
                            "Value": "07ce06bb-bf5a-4d8d-a639-2db96577c8b6.5edc343e-f218-51b6-bbe5-9b57038c1a9b"
                        },
                        {
                            "Key": "Name",
                            "Value": "dev-network-hector"
                        }
                    ],
                    "VpcId": "vpc-3e9e5e59"
                }
            }
        }
    }
},
"state": "BUSY",
"updated": 1479675576
}
```

There’s a lot of good detail in there and its easy to grep for different things you may need to know, like a VPC ID.

Since we no longer need Lisa’s environment, let’s terminate her VPC and all of it’s associated resources by killing the process using the fugue kill command. But since we’re taking a destructive step, let’s make sure that we know what will be deleted before any real operations are triggered. The dry-run option shows us everything that the Conductor will do when we terminate Lisa’s workload.

`fugue kill dev-lisa --dry-run`

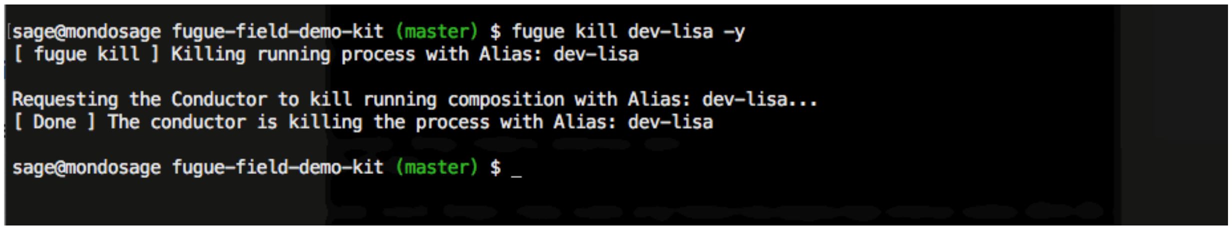


```
            "request_type": "aws.ec2.detach_internet_gateway"
        },
        {
            "guid": "9b567e78-2b5b-48ba-8c80-0667674df4c6.427db8da-50bc-5284-a38f-eadd236c1715",
            "params": {
                "SubnetId": "subnet-35ce2152"
            },
            "region": "us-west-2",
            "request_type": "aws.ec2.delete_subnet"
        },
        {
            "guid": "9b567e78-2b5b-48ba-8c80-0667674df4c6.fec6c182-4b53-568d-ab12-85f7a770556b",
            "params": {
                "SubnetId": "subnet-7902030f"
            },
            "region": "us-west-2",
            "request_type": "aws.ec2.delete_subnet"
        },
        {
            "guid": "9b567e78-2b5b-48ba-8c80-0667674df4c6.1224c310-6bcf-53db-8887-94fadbb4b3a0",
            "params": {
                "VpcId": "vpc-0e814c69"
            },
            "region": "us-west-2",
            "request_type": "aws.ec2.delete_vpc"
        },
        {
            "guid": "9b567e78-2b5b-48ba-8c80-0667674df4c6.e736595a-0041-57d5-bd0a-b42928b2ae42",
            "params": {
                "DhcpOptionsId": "dopt-3d004a59"
            },
            "region": "us-west-2",
            "request_type": "aws.ec2.delete_dhcp_options"
        }
    ]
}
sage@mondosage fugue-field-demo-kit (master) $ _
```

Each “request_type” in the output of the dry run shows us the specific resource that will be deleted.

Ok, now let’s proceed with the actual kill operation and terminate her VPC.

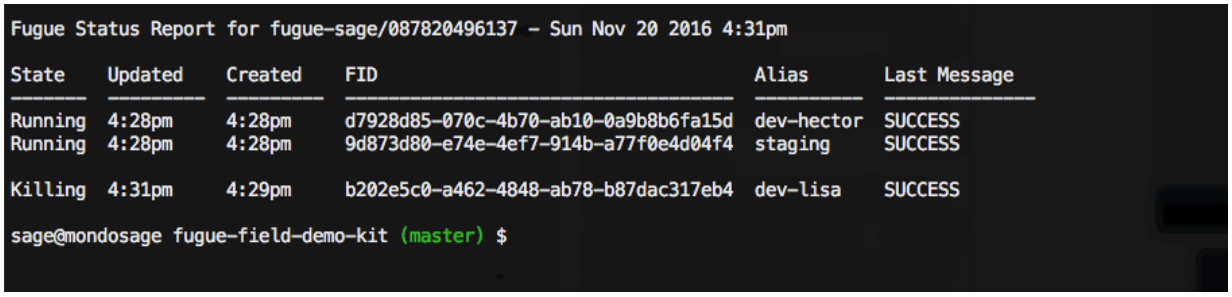
`fugue kill dev-lisa`



```
sage@mondosage fugue-field-demo-kit (master) $ fugue kill dev-lisa -y
[ fugue kill ] Killing running process with Alias: dev-lisa

Requesting the Conductor to kill running composition with Alias: dev-lisa...
[ Done ] The conductor is killing the process with Alias: dev-lisa

sage@mondosage fugue-field-demo-kit (master) $ _
```

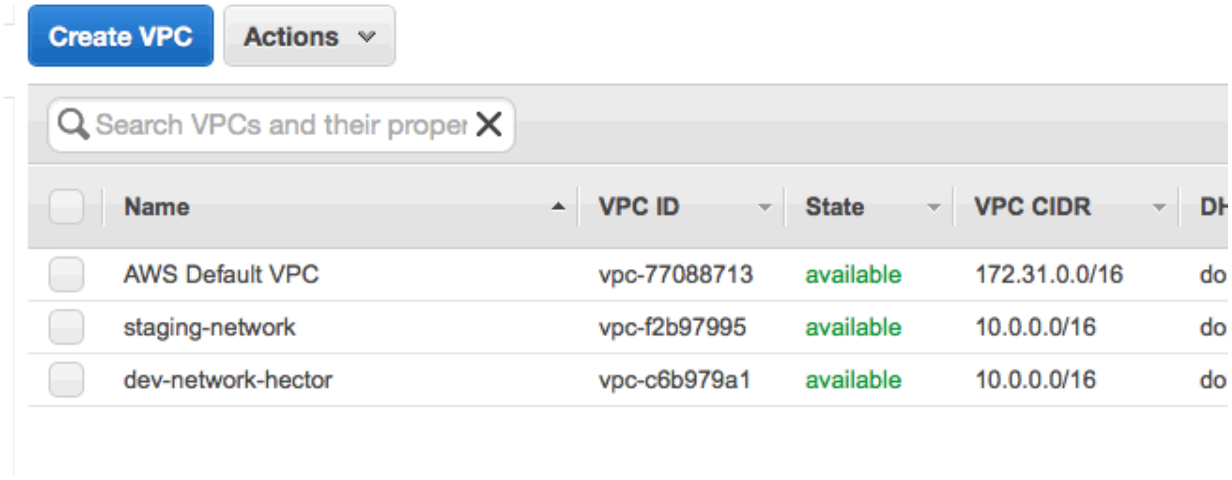
Ok. Looking at fugue status again we can see that her process’s state has changed to Killing.

`fugue status`



```
Fugue Status Report for fugue-sage/087820496137 - Sun Nov 20 2016 4:31pm

State    Updated    Created    FID                                   Alias       Last Message
-------  ---------  ---------  ------------------------------------  ----------  ------------
Running  4:28pm     4:28pm     d7928d85-070c-4b70-ab10-0a9b8b6fa15d  dev-hector  SUCCESS
Running  4:28pm     4:28pm     9d873d80-e74e-4ef7-914b-a77f0e4d04f4  staging     SUCCESS

Killing  4:31pm     4:29pm     b202e5c0-a462-4848-ab78-b87dac317eb4  dev-lisa    SUCCESS

sage@mondosage fugue-field-demo-kit (master) $
```

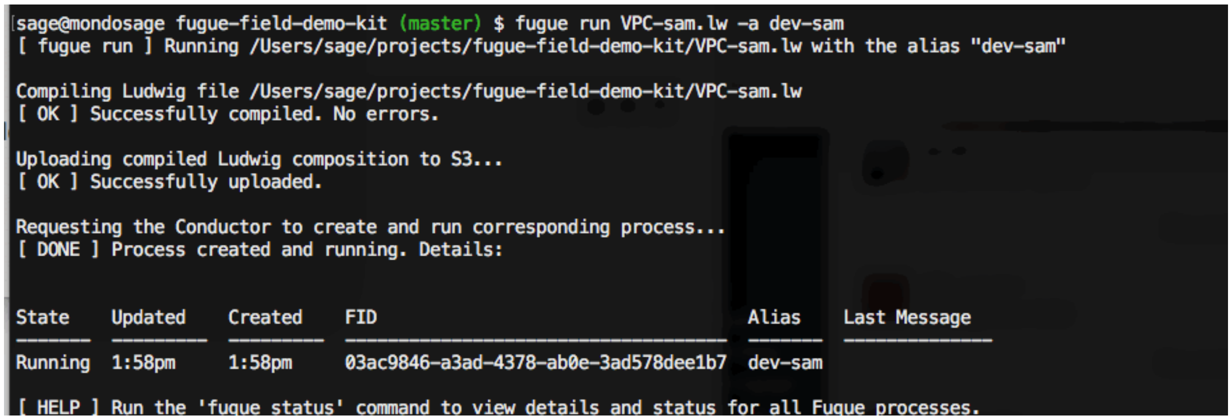
And when we refresh the web console, her network is now gone.



Name	VPC ID	State	VPC CIDR	DH
AWS Default VPC	vpc-77088713	available	172.31.0.0/16	do
staging-network	vpc-f2b97995	available	10.0.0.0/16	do
dev-network-hector	vpc-c6b979a1	available	10.0.0.0/16	do

Now, since Sam needs a new VPC, let’s spin his up. Here, we’ll use the fugue run command with his composition. We’ll talk about compositions in a few minutes.

`fugue run VPC-sam.lw -a dev-sam`

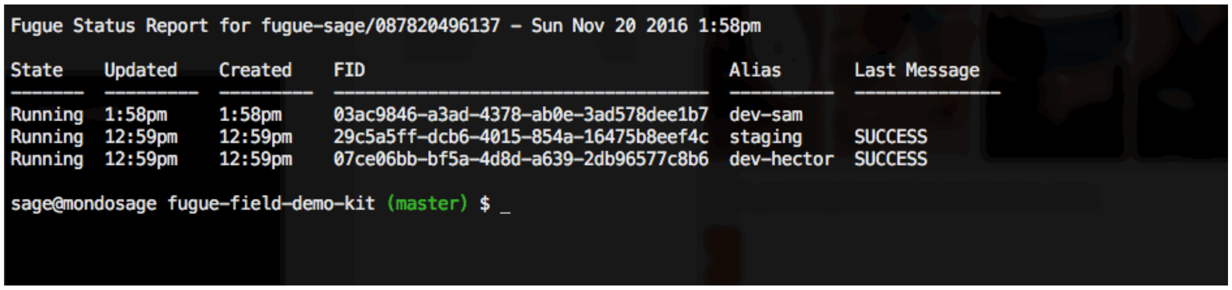


```
sage@mondosage fugue-field-demo-kit (master) $ fugue run VPC-sam.lw -a dev-sam
[ fugue run ] Running /Users/sage/projects/fugue-field-demo-kit/VPC-sam.lw with the alias "dev-sam"

Compiling Ludwig file /Users/sage/projects/fugue-field-demo-kit/VPC-sam.lw
[ OK ] Successfully compiled. No errors.

Uploading compiled Ludwig composition to S3...
[ OK ] Successfully uploaded.

Requesting the Conductor to create and run corresponding process...
[ DONE ] Process created and running. Details:

State    Updated    Created    FID                                   Alias     Last Message
-------  ---------  ---------  ------------------------------------  -------   ------------
Running  1:58pm     1:58pm     03ac9846-a3ad-4378-ab0e-3ad578dee1b7  dev-sam

[ HELP ] Run the 'fugue status' command to view details and status for all Fugue processes.
```

The composition for Sam's VPC is compiled and uploaded to the Conductor which will do the work of spinning up all the necessary VPC resources.

Now, when we type `fugue status` we'll see the new process added to the list.

`fugue status`



```
Fugue Status Report for fugue-sage/087820496137 - Sun Nov 20 2016 1:58pm

State    Updated   Created   FID                                   Alias       Last Message
-------  --------  --------  ------------------------------------  ----------  ------------
Running  1:58pm    1:58pm    03ac9846-a3ad-4378-ab0e-3ad578dee1b7  dev-sam
Running  12:59pm   12:59pm   29c5a5ff-dcb6-4015-854a-16475b8eef4c  staging     SUCCESS
Running  12:59pm   12:59pm   07ce06bb-bf5a-4d8d-a639-2db96577c8b6  dev-hector  SUCCESS

sage@mondosage fugue-field-demo-kit (master) $ _
```

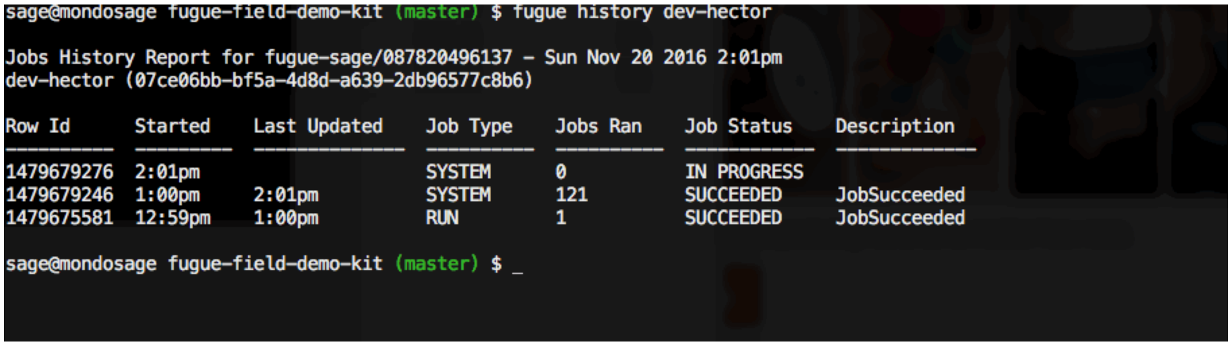
Now, each of these process is managed by the Conductor, as we've said. The Conductor has a few jobs to do, beyond just provisioning or de-provisioning resources.

Typing `fugue history <alias>` will show us different jobs that have been run by the Conductor. Some of them are triggered by our CLI commands, others are internal

System jobs.

Let’s see what’s been going on with the process we just kicked off for Sam’s VPC.

`fugue history dev-sam`



```
sage@mondosage fugue-field-demo-kit (master) $ fugue history dev-hector

Jobs History Report for fugue-sage/087820496137 - Sun Nov 20 2016 2:01pm
dev-hector (07ce06bb-bf5a-4d8d-a639-2db96577c8b6)

Row Id      Started    Last Updated    Job Type    Jobs Ran    Job Status     Description
----------  ---------  --------------  ----------  ----------  -------------  -------------
1479679276  2:01pm                     SYSTEM      0           IN PROGRESS
1479679246  1:00pm     2:01pm          SYSTEM      121         SUCCEEDED      JobSucceeded
1479675581  12:59pm    1:00pm          RUN         1           SUCCEEDED      JobSucceeded

sage@mondosage fugue-field-demo-kit (master) $ _
```

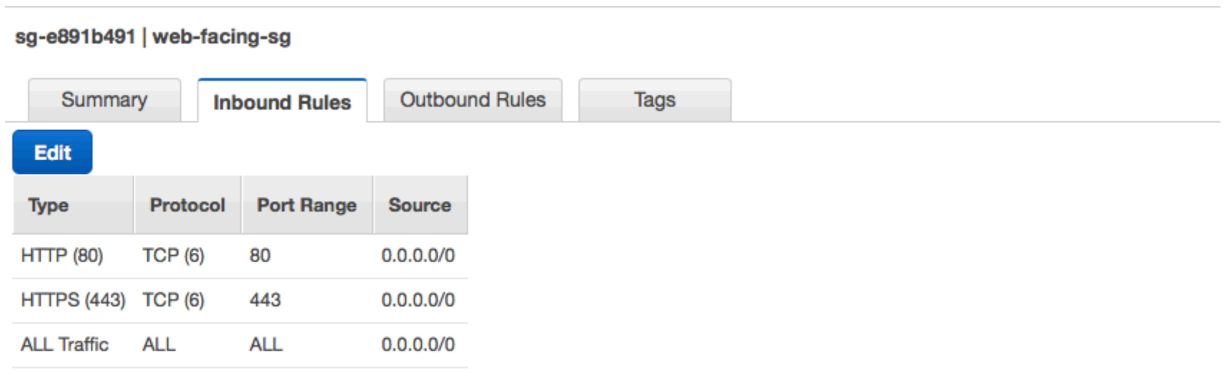
You see the first job that we ran at the bottom of the list, the Run operation.

Among the system jobs that the Conductor performs is enforcing the configuration state that’s been declared in the corresponding composition. It does this as a control loop, always inspecting the live environment and comparing it to the desired state, correcting any divergences.

Let’s take a look at how that works.

As we said during the intro, Hector is working on integrations. He’s been having some difficulty communicating with a third-party service, and so let’s say he decides to just open his VPC’s security group to all traffic from the world.

(go to security group web console for dev-sam-sg and add ALL Traffic from 0.0.0.0/0)


sg-e891b491 | web-facing-sg

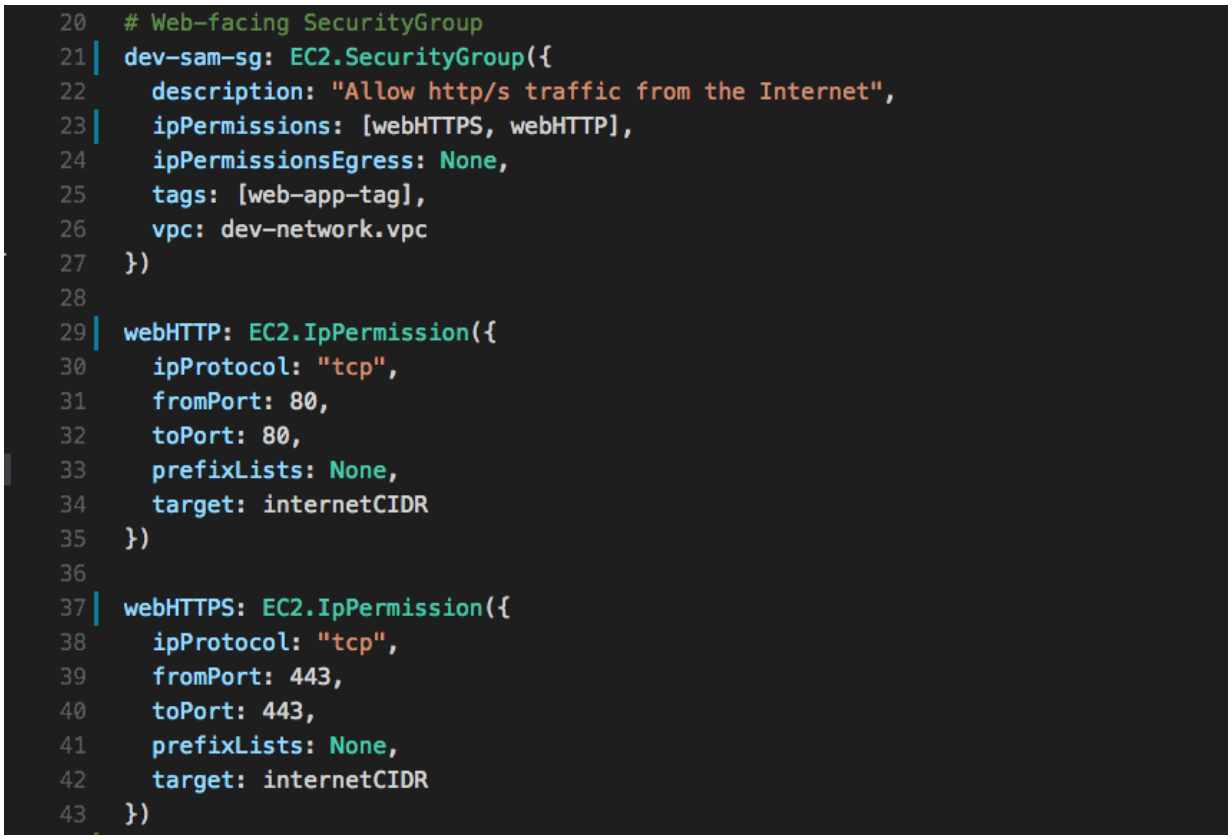
Type	Protocol	Port Range	Source
HTTP (80)	TCP (6)	80	0.0.0.0/0
HTTPS (443)	TCP (6)	443	0.0.0.0/0
ALL Traffic	ALL	ALL	0.0.0.0/0

Obviously this is a bad idea.

(ok, here we have to keep talking for about a minute while we wait for the conductor tick, so we'll introduce compositions)

Now let's take a look at our VPC composition.

We'll be exploring the composition further in a moment, but for now, let's take a look at how we configured our security group.

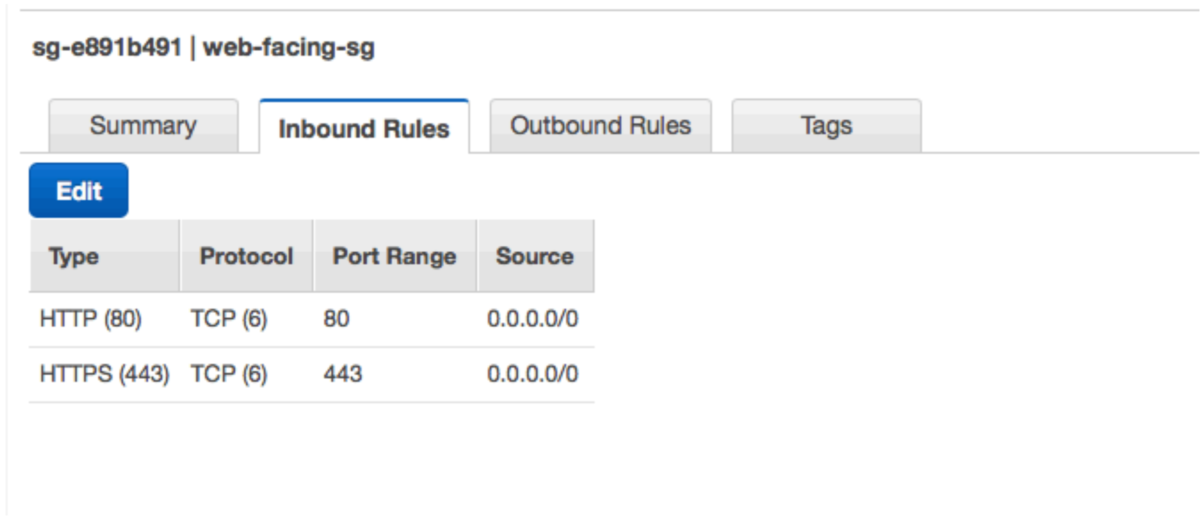


```
# Web-facing SecurityGroup
dev-sam-sg: EC2.SecurityGroup({
  description: "Allow http/s traffic from the Internet",
  ipPermissions: [webHTTPS, webHTTP],
  ipPermissionsEgress: None,
  tags: [web-app-tag],
  vpc: dev-network.vpc
})

webHTTP: EC2.IpPermission({
  ipProtocol: "tcp",
  fromPort: 80,
  toPort: 80,
  prefixLists: None,
  target: internetCIDR
})

webHTTPS: EC2.IpPermission({
  ipProtocol: "tcp",
  fromPort: 443,
  toPort: 443,
  prefixLists: None,
  target: internetCIDR
})
```

As you can see, compositions are declarative. Here we've defined a security group for our VPC, and we've defined two permission blocks, one for HTTP and another HTTPS.

When it was compiled and uploaded, this became what the Conductor uses as its source of truth. When it runs its inspection loop, it will detect the new rule that Hector added and remove it since it diverges from the composition.

(refresh SG console.. if the SSH rule is still there, "The conductor runs its loop every 30 seconds or so, so the rule should be removed in just a moment")

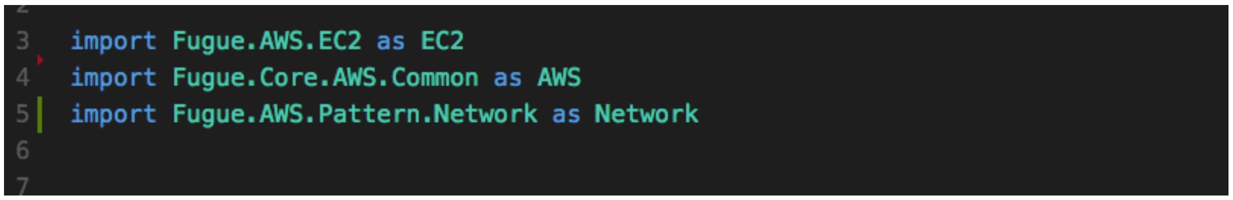


Type	Protocol	Port Range	Source
HTTP (80)	TCP (6)	80	0.0.0.0/0
HTTPS (443)	TCP (6)	443	0.0.0.0/0

And there you go! The rogue rule is gone and the security group is back to the desired state.

Now let's look a bit more at compositions. Fugue compositions are written in a declarative, statically-typed programming language called Ludwig.

At the top of the composition you can see some imported code. Ludwig uses a modular library system so we can create reusable code and make things very repeatable and straightforward for engineers working with AWS.



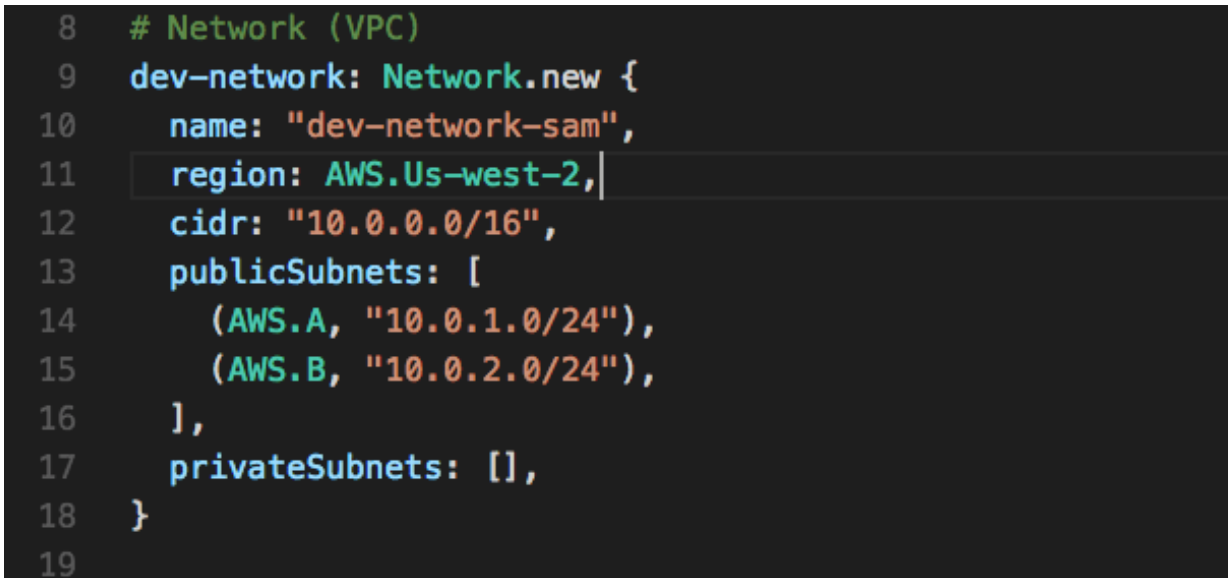
```
import Fugue.AWS.EC2 as EC2
import Fugue.Core.AWS.Common as AWS
import Fugue.AWS.Pattern.Network as Network
```

Here have three imports —

- the EC2 library is where the VPC and Security Group types are defined, so we need to import those to create the resources.

- The Common library let’s us use common things like regions and tags.

- The network pattern library is where we’ve defined how to create the underlying components required for a VPC, like the internet gateway and route tables and so on, so that developers can simply provide a few field values and have it all created for them automatically.



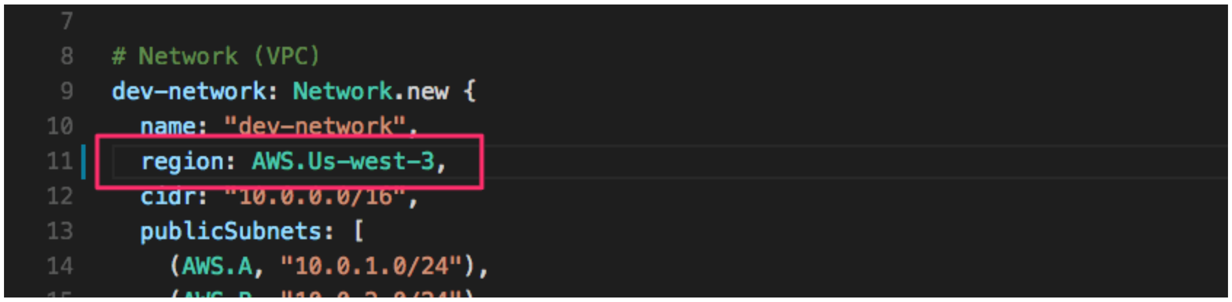
```
# Network (VPC)
dev-network: Network.new {
  name: "dev-network-sam",
  region: AWS.Us-west-2,
  cidr: "10.0.0.0/16",
  publicSubnets: [
    (AWS.A, "10.0.1.0/24"),
    (AWS.B, "10.0.2.0/24"),
  ],
  privateSubnets: [],
}
```

I mentioned that Ludwig is statically typed, and AWS resources are defined as types in Ludwig, which means we can check the configuration of them at compile time.

Let’s look at an example:

One simple example is to try to use a non-existent region. Maybe we accidentally type `us-west-3` as our region.

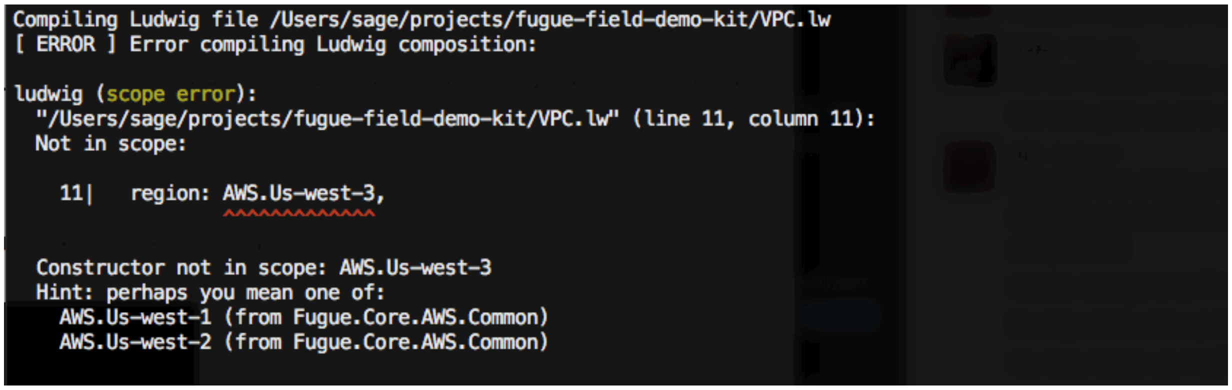
(edit composition to change region: to Us-west-3)



```
# Network (VPC)
dev-network: Network.new {
  name: "dev-network",
  region: AWS.Us-west-3,
  cidr: "10.0.0.0/16",
  publicSubnets: [
    (AWS.A, "10.0.1.0/24"),
```

If we try to `fugue run` this, the compiler will generate an error:

`fugue run VPC-sam.lw`



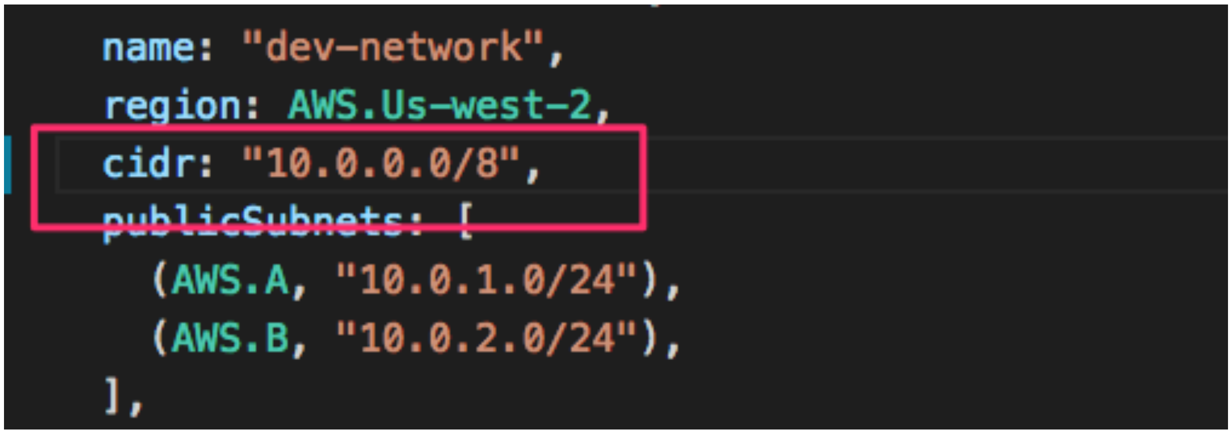
```
Compiling Ludwig file /Users/sage/projects/fugue-field-demo-kit/VPC.lw
[ ERROR ] Error compiling Ludwig composition:

ludwig (scope error):
  "/Users/sage/projects/fugue-field-demo-kit/VPC.lw" (line 11, column 11):
  Not in scope:

    11|   region: AWS.Us-west-3,
                  ^^^^^^^^^^^^^

  Constructor not in scope: AWS.Us-west-3
  Hint: perhaps you mean one of:
    AWS.Us-west-1 (from Fugue.Core.AWS.Common)
    AWS.Us-west-2 (from Fugue.Core.AWS.Common)
```

That’s a simple example, but it illustrates the behavior we want.

A more realistic example might be using a CIDR range that is not allowable in AWS, like /8.

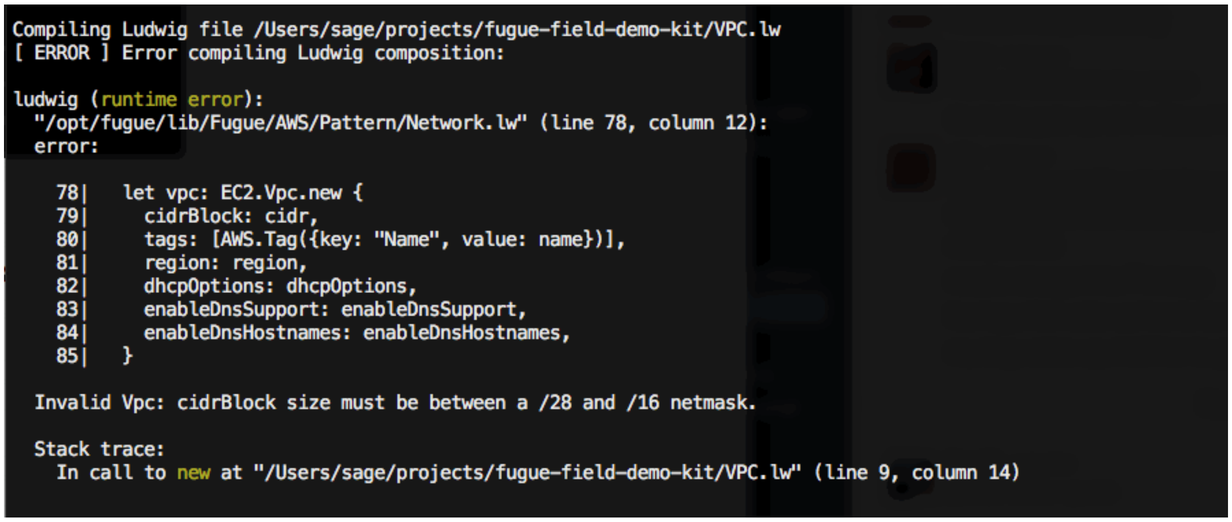
Let’s try that:

(fix the region back to us-west-2, and edit the CIDR block with 10.0.0.0/8)



```
  name: "dev-network",
  region: AWS.Us-west-2,
  cidr: "10.0.0.0/8",
  publicSubnets: [
    (AWS.A, "10.0.1.0/24"),
    (AWS.B, "10.0.2.0/24"),
  ],
```

Now we have another error, this one based on an AWS constraint. The compiler shows us the relevant code and stack trace so we can troubleshoot it easily if we have to, and also a meaningful error message so we can fix it right away.



```
Compiling Ludwig file /Users/sage/projects/fugue-field-demo-kit/VPC.lw
[ ERROR ] Error compiling Ludwig composition:

ludwig (runtime error):
  "/opt/fugue/lib/Fugue/AWS/Pattern/Network.lw" (line 78, column 12):
  error:

    78|   let vpc: EC2.Vpc.new {
    79|     cidrBlock: cidr,
    80|     tags: [AWS.Tag({key: "Name", value: name})],
    81|     region: region,
    82|     dhcpOptions: dhcpOptions,
    83|     enableDnsSupport: enableDnsSupport,
    84|     enableDnsHostnames: enableDnsHostnames,
    85|   }

  Invalid Vpc: cidrBlock size must be between a /28 and /16 netmask.

  Stack trace:
    In call to new at "/Users/sage/projects/fugue-field-demo-kit/VPC.lw" (line 9, column 14)
```

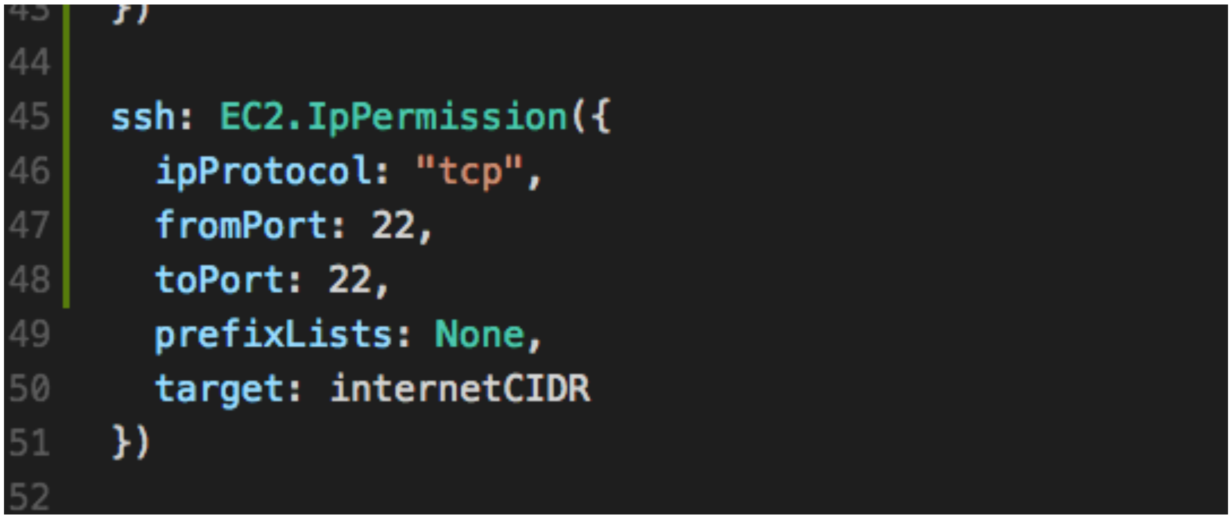
These kinds of AWS-specific validations are baked into Fugue so you can get a little help making sure things will work before you instantiate any infrastructure.

That covers compile time validation.

Now that we've set up Sam's new VPC, we found out he's going to need SSH access to some instances he's working with.

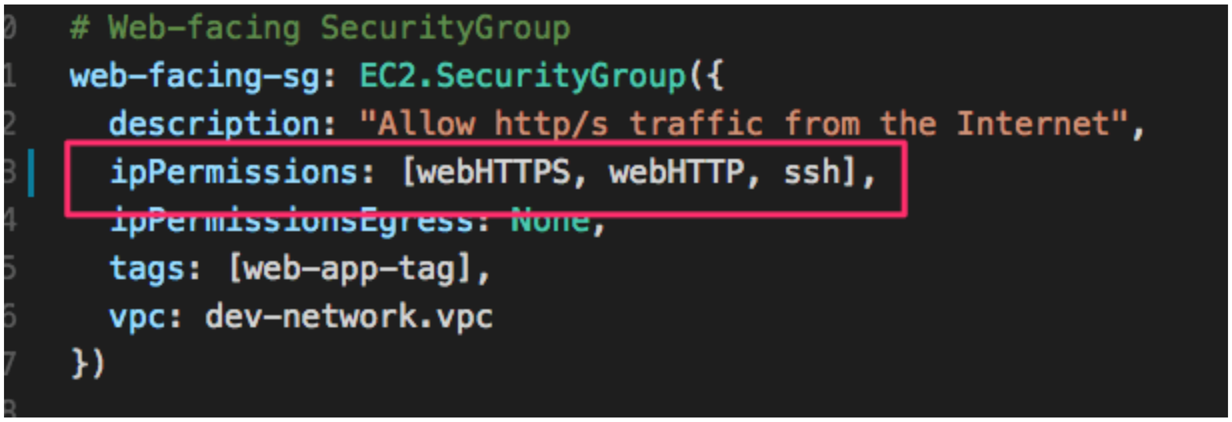
For that we need to add a new permission block to his composition and run an update.

First, let's add the new block:



```
})

ssh: EC2.IpPermission({
  ipProtocol: "tcp",
  fromPort: 22,
  toPort: 22,
  prefixLists: None,
  target: internetCIDR
})

```

and add the new rule to our SG permissions list:



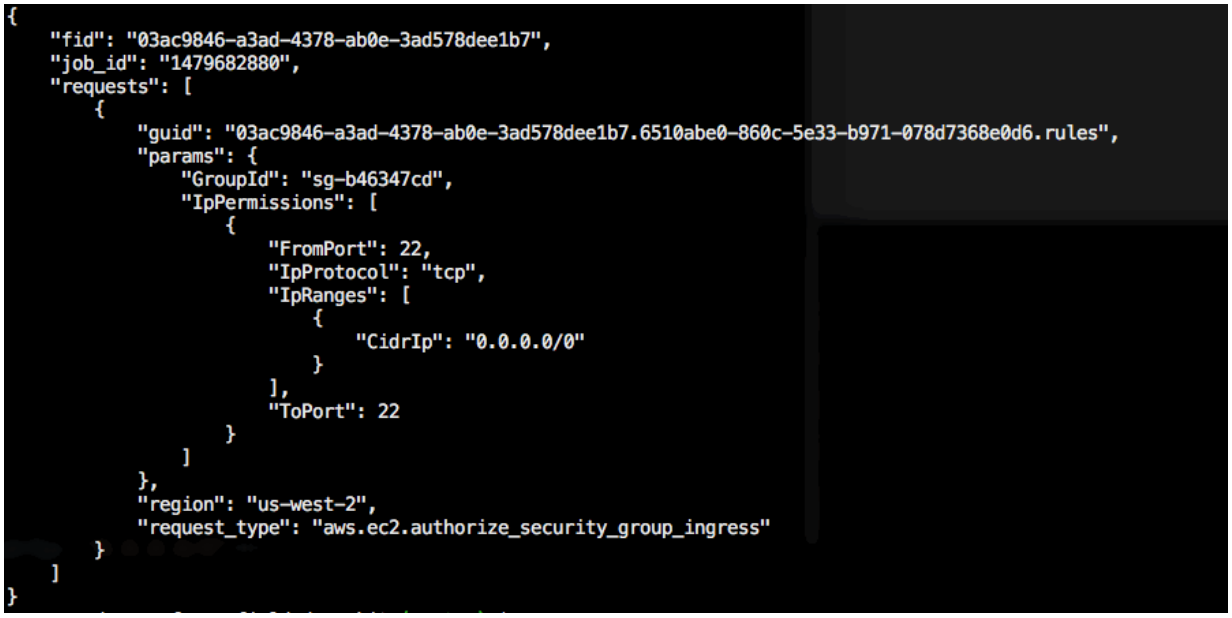
```
# Web-facing SecurityGroup
web-facing-sg: EC2.SecurityGroup({
  description: "Allow http/s traffic from the Internet",
  ipPermissions: [webHTTPS, webHTTP, ssh],
  ipPermissionsEgress: None,
  tags: [web-app-tag],
  vpc: dev-network.vpc
})
```

Let’s save the updated composition and return to the CLI.

When you run an update like this, it’s usually a good idea to see what you’re going to change before you take any action, like we did earlier when we retired Lisa’s VPC..

With fugue, we use the —dry-run option with the update command to verify that we really want to do this.

`fugue update dev-sam VPC-sam.lw  - -dry-run`

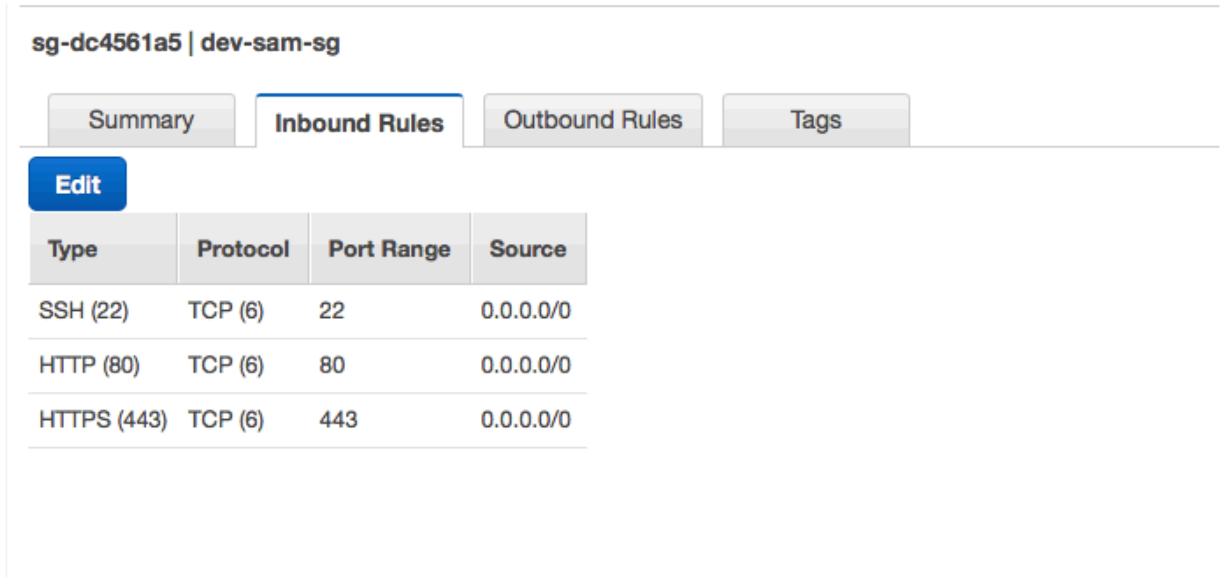


```
{
    "fid": "03ac9846-a3ad-4378-ab0e-3ad578dee1b7",
    "job_id": "1479682880",
    "requests": [
        {
            "guid": "03ac9846-a3ad-4378-ab0e-3ad578dee1b7.6510abe0-860c-5e33-b971-078d7368e0d6.rules",
            "params": {
                "GroupId": "sg-b46347cd",
                "IpPermissions": [
                    {
                        "FromPort": 22,
                        "IpProtocol": "tcp",
                        "IpRanges": [
                            {
                                "CidrIp": "0.0.0.0/0"
                            }
                        ],
                        "ToPort": 22
                    }
                ]
            },
            "region": "us-west-2",
            "request_type": "aws.ec2.authorize_security_group_ingress"
        }
    ]
}
```

We can see that Fugue is going to add an ingress rule for port 22, which is exactly what we want, so now we can re-run the update without the dry-run so it really happens.

`fugue update dev-sam VPC-sam.lw`

Let’s go back to the web console for Sam’s security group and we should see the SSH port now.



Type	Protocol	Port Range	Source
SSH (22)	TCP (6)	22	0.0.0.0/0
HTTP (80)	TCP (6)	80	0.0.0.0/0
HTTPS (443)	TCP (6)	443	0.0.0.0/0

That concludes the main features of our demo today.

Before we wrap up, though, I’d like to give you a glimpse of something new.

As you’ve seen, Fugue compositions represent what’s actually running in our AWS workloads. Our VPC example was pretty simple, and in many cases workloads can be quite a lot of resources with complex relationships.

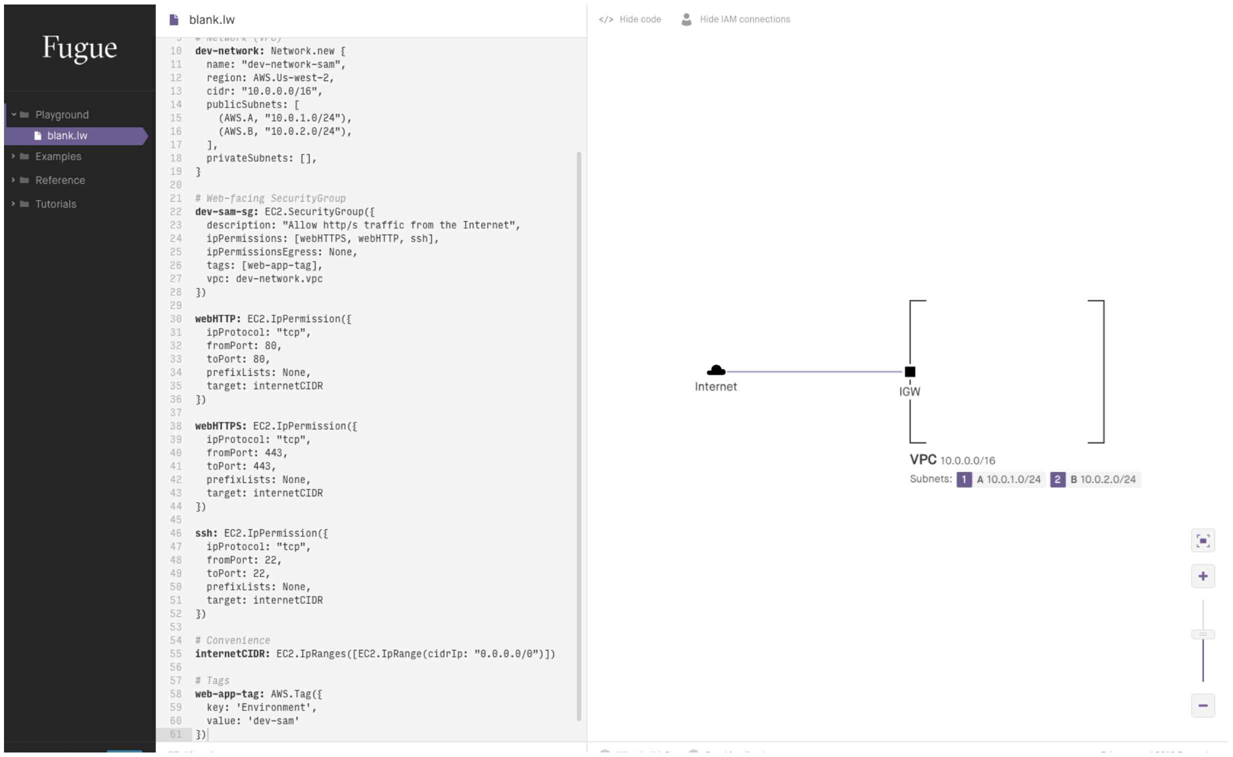
As a Fugue user, you want to see what your code is going to do, and be able to easily visualize that. We're working on new graphical visualizer for exactly those reasons.

Check it out:

(go to https://beta-playground.fugue.co)

The visualizer starts us off with a blank file. Let's paste our VPC code into it and click Run.

(copy/paste code into blank.lw)

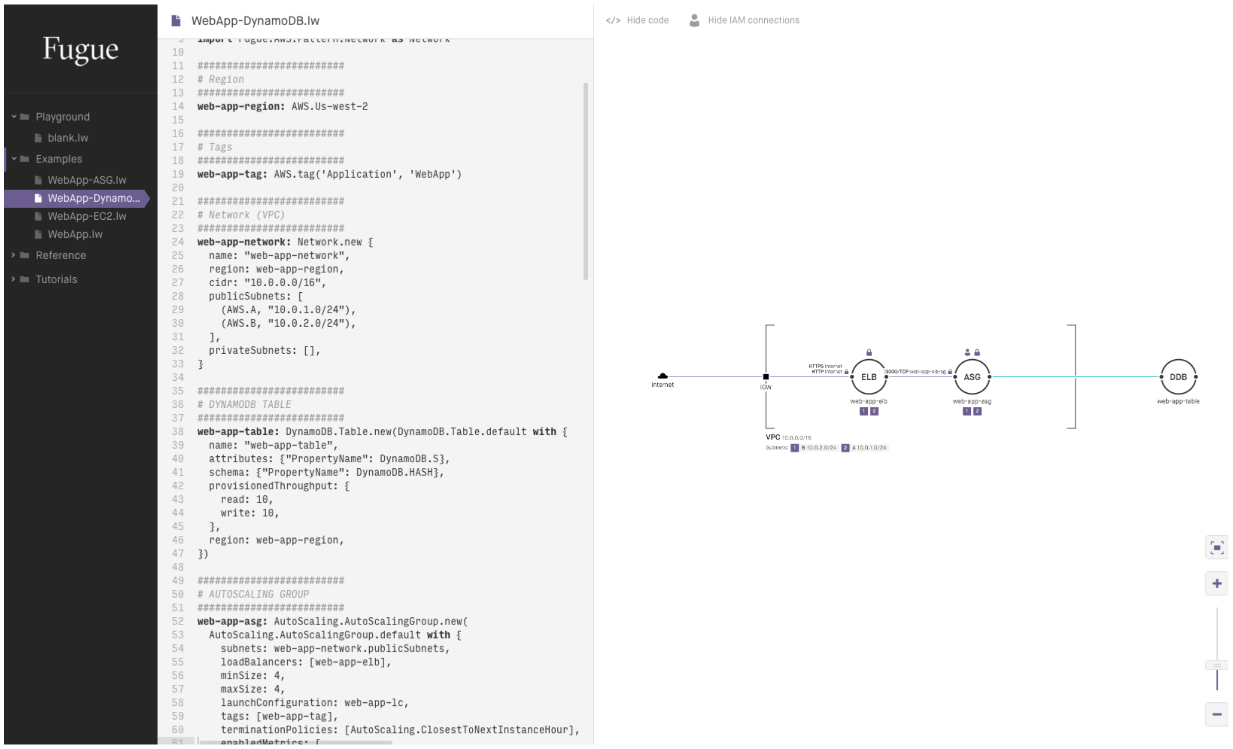


```
dev-network: Network.new {
  name: "dev-network-sam",
  region: AWS.Us-west-2,
  cidr: "10.0.0.0/16",
  publicSubnets: [
    (AWS.A, "10.0.1.0/24"),
    (AWS.B, "10.0.2.0/24"),
  ],
  privateSubnets: [],
}

# Web-facing SecurityGroup
dev-sam-sg: EC2.SecurityGroup({
  description: "Allow http/s traffic from the Internet",
  ipPermissions: [webHTTPS, webHTTP, ssh],
  ipPermissionsEgress: None,
  tags: [web-app-tag],
  vpc: dev-network.vpc
})

webHTTP: EC2.IpPermission({
  ipProtocol: "tcp",
  fromPort: 80,
  toPort: 80,
  prefixLists: None,
  target: internetCIDR
})

webHTTPS: EC2.IpPermission({
  ipProtocol: "tcp",
  fromPort: 443,
  toPort: 443,
  prefixLists: None,
  target: internetCIDR
})

ssh: EC2.IpPermission({
  ipProtocol: "tcp",
  fromPort: 22,
  toPort: 22,
  prefixLists: None,
  target: internetCIDR
})

# Convenience
internetCIDR: EC2.IpRanges([EC2.IpRange(cidrIp: "0.0.0.0/0")])

# Tags
web-app-tag: AWS.Tag({
  key: 'Environment',
  value: 'dev-sam'
})
```

Our VPC is pretty simple and we just see the internet gateway, the VPC brackets, and our two subnets in the AZs.

A richer example is the WebApp-DynamoDB.lw composition, which starts with our VPC and adds an ELB, an ASG, and a DDB backend.

(select WebApp-DynamoDB.lw under Examples)



```
import Fugue.AWS.Pattern.Network as Network

#########################
# Region
#########################
web-app-region: AWS.Us-west-2

#########################
# Tags
#########################
web-app-tag: AWS.tag('Application', 'WebApp')

#########################
# Network (VPC)
#########################
web-app-network: Network.new {
  name: "web-app-network",
  region: web-app-region,
  cidr: "10.0.0.0/16",
  publicSubnets: [
    (AWS.A, "10.0.1.0/24"),
    (AWS.B, "10.0.2.0/24"),
  ],
  privateSubnets: [],
}

#########################
# DYNAMODB TABLE
#########################
web-app-table: DynamoDB.Table.new(DynamoDB.Table.default with {
  name: "web-app-table",
  attributes: {"PropertyName": DynamoDB.S},
  schema: {"PropertyName": DynamoDB.HASH},
  provisionedThroughput: {
    read: 10,
    write: 10,
  },
  region: web-app-region,
})

#########################
# AUTOSCALING GROUP
#########################
web-app-asg: AutoScaling.AutoScalingGroup.new(
  AutoScaling.AutoScalingGroup.default with {
    subnets: web-app-network.publicSubnets,
    loadBalancers: [web-app-elb],
    minSize: 4,
    maxSize: 4,
    launchConfiguration: web-app-lc,
    tags: [web-app-tag],
    terminationPolicies: [AutoScaling.ClosestToNextInstanceHour],
    enabledMetrics: [
```

Its a super cool way of seeing what’s going on in your composition and therefore being managed by the conductor.

Ok, that’s our presentation. Be sure to sign up for access at fugue.co/free and enter to win the Moog synth device. If you’d like a one-on-one session just let one of our booth staffers know and we’ll set it up.

Now I’m happy to take your questions.