

Performance
testing meets
The Cloud

I Tea & Coffee

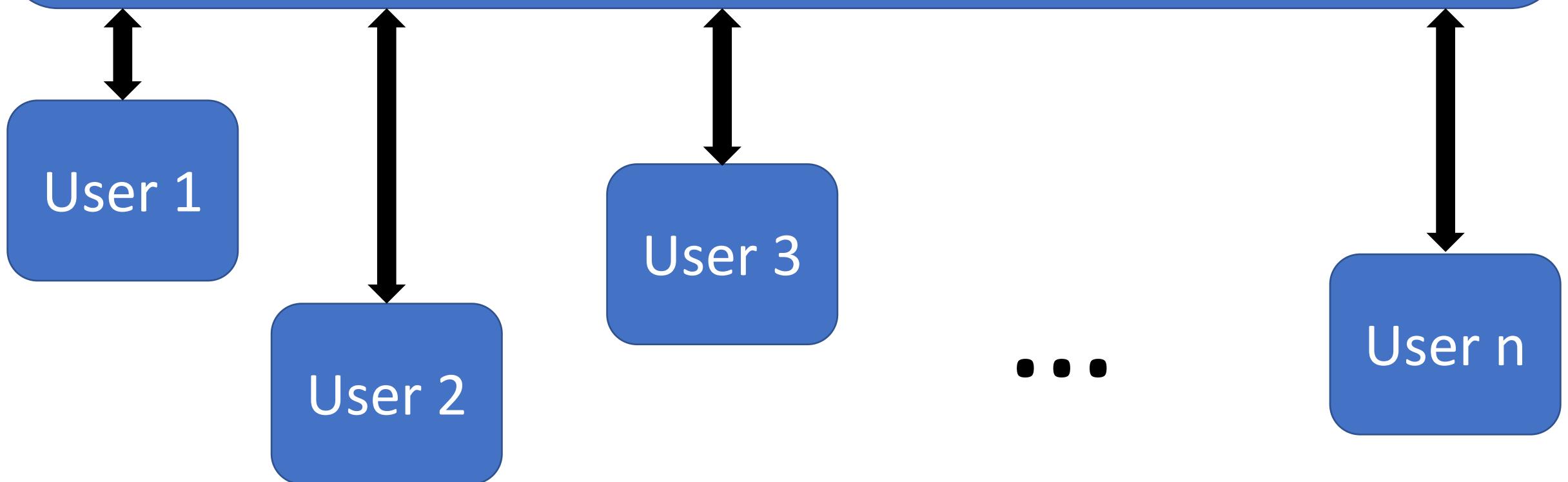


What is Performance Testing?

Wiki:
a **testing practice** performed to
determine **how a system performs**
in terms of **responsiveness and stability**
under a particular workload

https://en.wikipedia.org/wiki/Software_performance_testing

Server (system under load)



Performance testing: best practices

- Users are different - i.e. sending unique (not hardcoded) requests
- Users are geographically distributed – so should be the test
- Performance testing should be done regularly

Emulate real users - JMeter

JMeter interface showing a Test Plan and results.

Test Plan:

- Thread Group
 - 1 Forecast Table - GetCityTableForecast
 - 2 Forecast Comparison - GetForecastComparison
 - 3 Hourly Forecast - GetHourlyForecast
 - 4 Weighted Forecast - ISO/Country - GetModelForecast
 - 5 Weighted Forecast - US Degree Days - GetModelDDForecast
 - 6 Weighted Forecast - US Degree Days - GetWeightedDegreeDays
 - 7 Day Forecast Graphics - GetForecastGraphics
 - 8 Day Forecast Headlines - GetForecastDiscussions
 - 9 Seasonal Forecast Graphics - GetSeasonalForecastGraphics
 - 10 Solar/Wind Power Forecasts - GetWindSolarForecast
 - 11 Historical Observations - GetHistoricalObservations
 - 12 Premium Weather - GetForecastAnalysis
 - 13 Premium Weather - GetTeleconnectionData
 - 14 Load - Hourly Forecasts - GetHourlyLoadData
 - 15 Load - Daily Forecasts - GetDailyLoadData
 - 16 Load - Observations - GetLoadObsData
 - 17 Natural Gas Demand Forecasts - GetModelBCFForecast
 - 18 Natural Gas Reports - GetBCFDocument
 - 19 WindCast IQ - Hourly Wind Power Forecasts - GetWindcastIQ
- HTTP Request Defaults
- View Results Tree
- Summary Report
- HTTP Request
- Notes
- WorkBench

View Results Tree:

Name: View Results Tree
Comments:
Write results to file / Read from file
Filename Browse... Log/Display Only: Errors Successes
Search: Case sensitive Regular exp.

Sampler result:

WSI ISO Region Weighted Forecast - Forecast Updated Jun 3 2018 1022 UTC
BELGIUM
,Min Temp, Max Temp, Min Temp Diff, Max Temp Diff, Min Temp DFN, Max Temp DFN, Average Temp DFN,
Day 1-6/3/2018,12.7,25,-0.6,2.9,2.5,5.5,4,
Day 2-6/4/2018,13.8,25.1,1.7,0.2,3.5,5.5,4.4,
Day 3-6/5/2018,13.3,24.5,0.3,-0.6,3.4,4.8,3.6,
Day 4-6/6/2018,12.9,27.8,0.3,3.6,2.4,8.1,4.8,
Day 5-6/7/2018,15.5,24.6,3.2,-1.4,5.4,4.8,5,
Day 6-6/8/2018,15,24.4,1.6,-1.7,4.4,4.6,4.3,
Day 7-6/9/2018,14.5,23.9,0.9,-2.1,3.9,3.9,3.5,
Day 8-6/10/2018,13.9,23.8,0.5,-2.5,3.1,3.8,3.1,
Day 9-6/11/2018,14.6,21.3,0.2,-1.9,3.8,1.2,2.6,
Day 10-6/12/2018,12.9,20.8,-0.4,-3.7,2,0.6,1.5,
Day 11-6/13/2018,12.5,21,-1.5,-0.9,1.5,0.7,1,
Day 12-6/14/2018,13.1,20.6,0.1,-2.2,2,0.3,1,
Day 13-6/15/2018,13,23.9,-0.1,0.8,1.9,3.5,2.1,
Day 14-6/16/2018,14.7,24.9,1.1,0.9,3.4,4.4,3.6,
Day 15-6/17/2018,16.3,24.4,NA,NA,4.9,3.8,4.2,
CZECH REPUBLIC

Request:

Search: Find Case sensitive Regular exp.

Response data:

Scrolled content showing various weather forecast details for Belgium and the Czech Republic.

Scroll automatically?

JMeter Pros:

- Easy to start with
- Collects performance metrics and generates pretty reports
- Scripts to generate unique requests on the go (Groovy, BeanShell, etc.)
- Active community and plenty of extensions for non-trivial tasks
- Allows distributed testing - *important*

However – when JMeter run “single handed”:

- No geographical diversity – all requests coming from the same region (throughput limitation)
- Limited abilities of a single instance to generate load (memory issues)

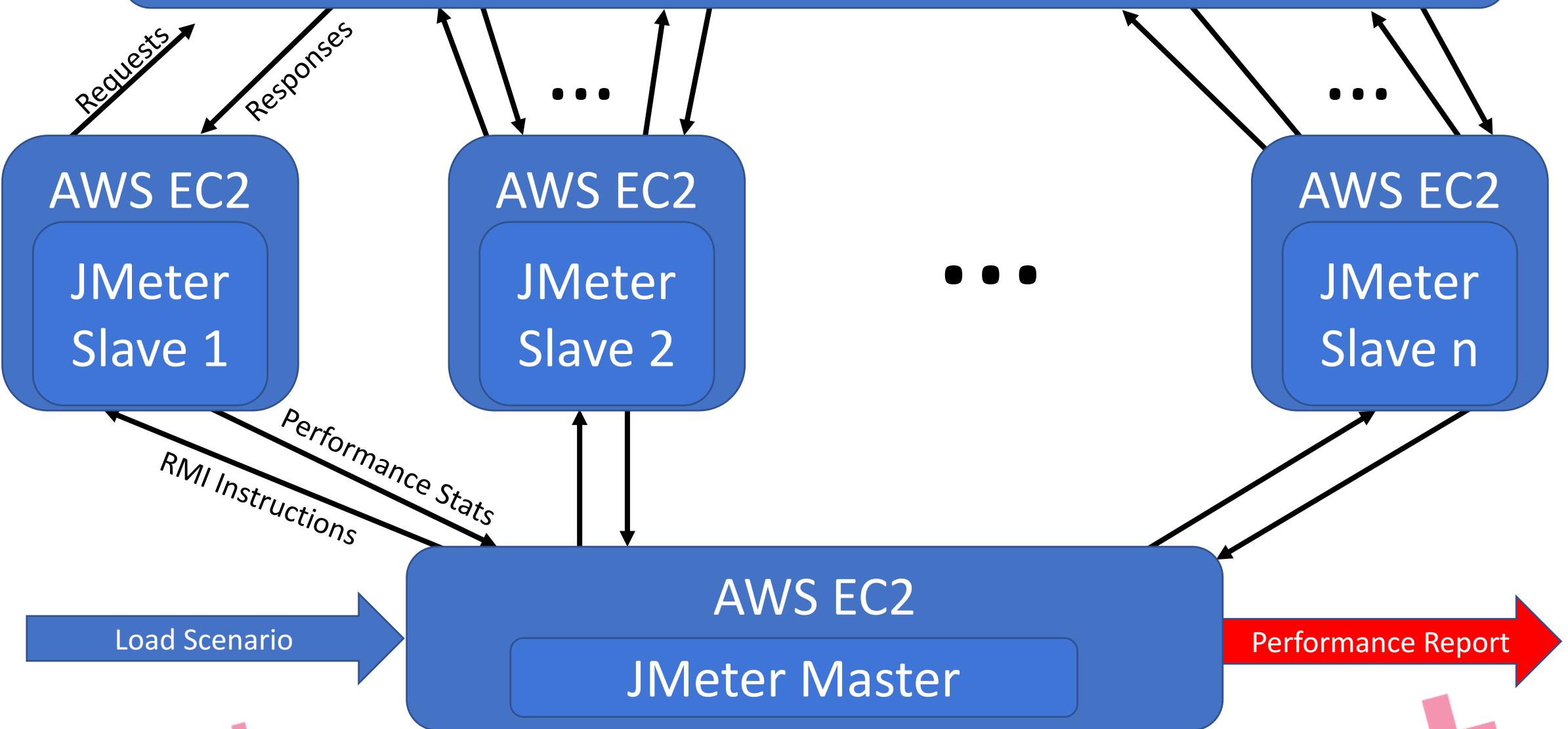
Thus:

running tests with a single JMeter instance is not a proper performance testing

Solution:

1. Set up several JMeter instances in different areas (Amazon Web Services might help)
2. Run performance tests in distributed mode

Server (system under load)



Setting up an AWS EC2 instance with JMeter:

- Set up Virtual Private Cloud;
- Set up Subnet;
- Set up Internet Gateway;
- Set up Route Table and Association;
- Set up Security Group;
- Add Key Pair;
- Start Instance with public IP;
- Install and configure JMeter (open ports, get IP).

Do that as many times as many instances you want.

Run distributed JMeter test in console:

```
jmeter -n -t load_script_name.jmx  
-R 54.172.38.36,...,18.144.50.21 -l report_name.jtl
```

JMeter Dockerized:

Ubuntu + Java

JMeter Master
(Manages Slaves):

- Port 60000 exposed

Ubuntu + Java

JMeter Slave
(Generates Load):
- Port 1099 exposed
- Port 50000 exposed

Dockerized JMeter on AWS:

AWS EC2:
Network, Docker

Ubuntu + Java

JMeter Master
(Manages Slaves):
- Port 60000 exposed

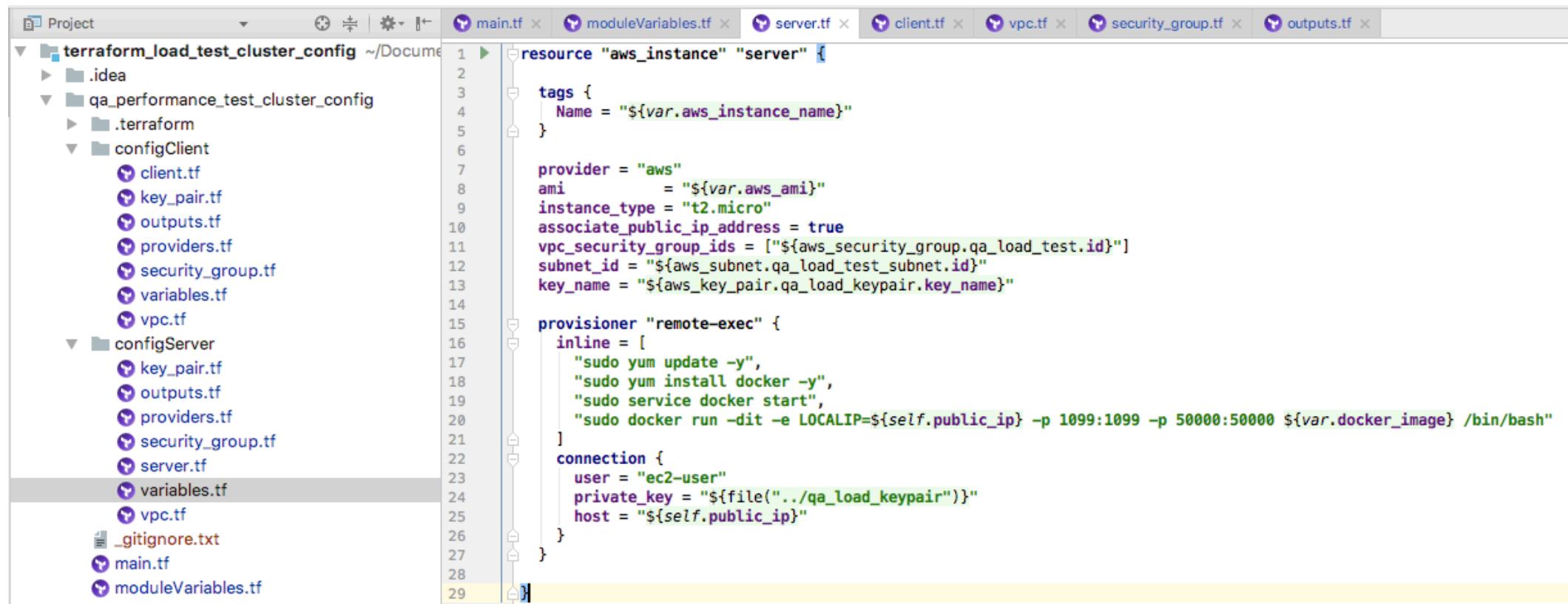
AWS EC2:
Network, Docker

Ubuntu + Java

JMeter Slave
(Generates Load):
- Ports 1099, 50000 exposed

Take automation one step further?

Configure the AWS infrastructure automatically with Terraform:



The screenshot shows a software interface for managing a Terraform project. On the left, a tree view displays the project structure:

- terraformer_load_test_cluster_config** (~/Documents)
- .idea
- qa_performance_test_cluster_config**
- .terraform
- configClient**
- client.tf
- key_pair.tf
- outputs.tf
- providers.tf
- security_group.tf
- variables.tf
- vpc.tf
- configServer**
- key_pair.tf
- outputs.tf
- providers.tf
- security_group.tf
- server.tf
- variables.tf
- vpc.tf
- _gitignore.txt
- main.tf
- moduleVariables.tf

The main window contains the **main.tf** file content:

```
resource "aws_instance" "server" {  
  tags {  
    Name = "${var.aws_instance_name}"  
  }  
  
  provider = "aws"  
  ami        = "${var.aws_ami}"  
  instance_type = "t2.micro"  
  associate_public_ip_address = true  
  vpc_security_group_ids = ["${aws_security_group.qa_load_test.id}"]  
  subnet_id = "${aws_subnet.qa_load_test_subnet.id}"  
  key_name = "${aws_key_pair.qa_load_keypair.key_name}"  
  
  provisioner "remote-exec" {  
    inline = [  
      "sudo yum update -y",  
      "sudo yum install docker -y",  
      "sudo service docker start",  
      "sudo docker run -dit -e LOCALIP=${self.public_ip} -p 1099:1099 -p 50000:50000 ${var.docker_image} /bin/bash"  
    ]  
    connection {  
      user = "ec2-user"  
      private_key = "${file("../qa_load_keypair")}"  
      host = "${self.public_ip}"  
    }  
  }  
}
```

Before you run the whole thing,

make sure to have:

- Valid AWS credentials in `~/.aws/credentials` file
- Git, Terraform installed

In the command line,

- clone the project:

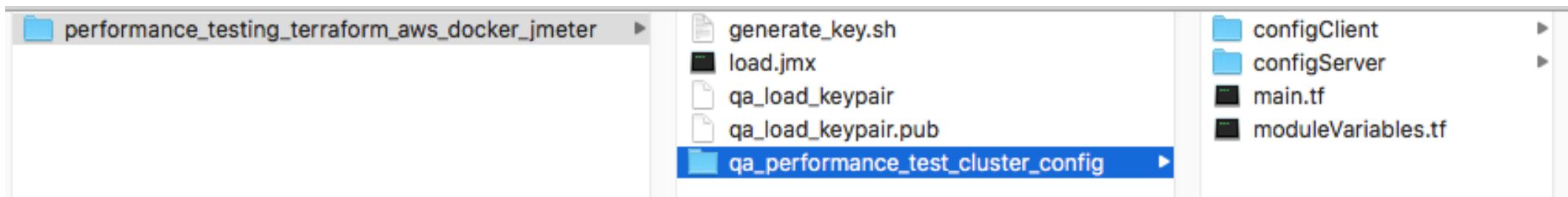
git clone https://github.com/artie-sh/performance_testing_terraform_aws_docker_jmeter.git

- enter the project folder:

cd performance_testing_terraform_aws_docker_jmeter

- generate ssh keys:

./generate_key.sh



- enter the terraform folder:

cd qa_performance_test_cluster_config

- initialize modules:

terraform init

```
Initializing modules...
- module.ireland
  Getting source "./configClient"
- module.n_virginia
  Getting source "./configServer"
- module.ohio
  Getting source "./configServer"
- module.n_california
  Getting source "./configServer"
- module.oregon
  Getting source "./configServer"
```

- plan the output:

terraform plan -out plan.output

```
+ module.oregon.aws_vpc.qa_load_test_vpc
  id:                                <computed>
  assign_generated_ipv6_cidr_block:    "false"
  cidr_block:                         "10.0.0.0/16"
  default_network_acl_id:            <computed>
  default_route_table_id:             <computed>
  default_security_group_id:          <computed>
  dhcp_options_id:                   <computed>
  enable_classiclink:                <computed>
  enable_classiclink_dns_support:    <computed>
  enable_dns_hostnames:              <computed>
  enable_dns_support:                 "true"
  instance_tenancy:                  <computed>
  ipv6_association_id:               <computed>
  ipv6_cidr_block:                   <computed>
  main_route_table_id:               <computed>
  tags.%:                            "1"
  tags.Name:                          "qa_load_test_vpc"
```

Plan: 40 to add, 0 to change, 0 to destroy.

This plan was saved to: plan.output

To perform exactly these actions, run the following command to apply:
terraform apply "plan.output"

- apply the plan:

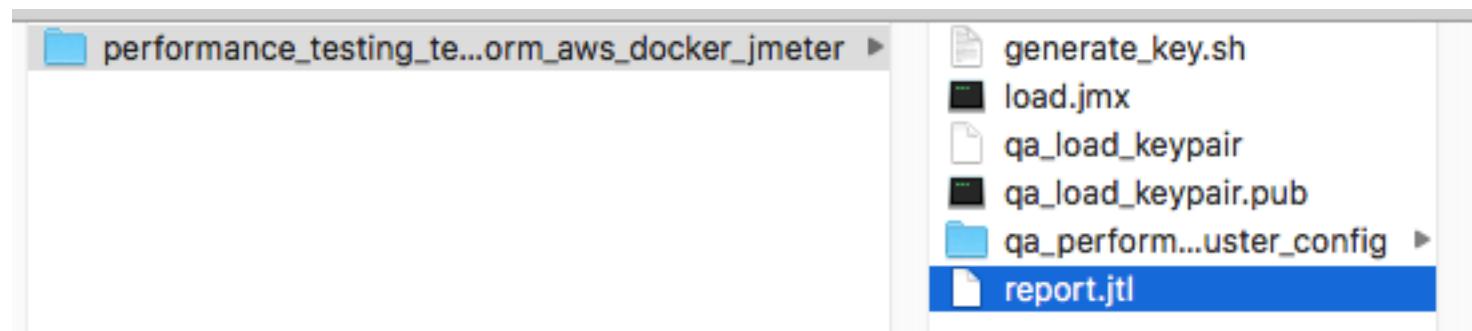
terraform apply plan.out

```
module.ireland.aws_instance.client (remote-exec): Creating summariser <summary>
module.ireland.aws_instance.client (remote-exec): Created the tree successfully using load.jmx
module.ireland.aws_instance.client (remote-exec): Configuring remote engine: 34.228.23.191
module.ireland.aws_instance.client (remote-exec): Configuring remote engine: 13.59.45.111
module.ireland.aws_instance.client (remote-exec): Configuring remote engine: 13.57.26.241
module.ireland.aws_instance.client (remote-exec): Configuring remote engine: 54.245.130.102
module.ireland.aws_instance.client: Still creating... (2m10s elapsed)
module.ireland.aws_instance.client (remote-exec): Starting remote engines
module.ireland.aws_instance.client (remote-exec): Starting the test @ Sun Jun  3 15:45:19 UTC 2018 (1528040719593)
module.ireland.aws_instance.client (remote-exec): Remote engines have been started
module.ireland.aws_instance.client (remote-exec): Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
module.ireland.aws_instance.client: Still creating... (2m20s elapsed)
module.ireland.aws_instance.client (remote-exec): summary +      4 in 00:00:06 =    0.7/s Avg:  1272
  Min:  777 Max: 2269 Err:    0 (0.00%) Active: 4 Started: 3 Finished: 0
module.ireland.aws_instance.client: Still creating... (2m30s elapsed)
module.ireland.aws_instance.client: Still creating... (2m40s elapsed)
module.ireland.aws_instance.client: Still creating... (2m50s elapsed)
module.ireland.aws_instance.client (remote-exec): summary +   401 in 00:00:35 =   11.4/s Avg:  198
  Min:   10 Max: 2863 Err:    0 (0.00%) Active: 4 Started: 3 Finished: 0
module.ireland.aws_instance.client (remote-exec): summary =   405 in 00:00:41 =   9.8/s Avg:  209
  Min:   10 Max: 2863 Err:    0 (0.00%)
```

- agree to accept the report file:
yes + enter

```
module.ireland.aws_instance.client: Provisioning with 'local-exec'...
module.ireland.aws_instance.client (local-exec): Executing: [/bin/sh "-c" "so
The authenticity of host '34.245.212.43 (34.245.212.43)' can't be established.
ECDSA key fingerprint is SHA256:way0ldyE41fbPKVVAzr5wMozj/P0SUURBHkxG1Mtqp4.
Are you sure you want to continue connecting (yes/no)? yes
```

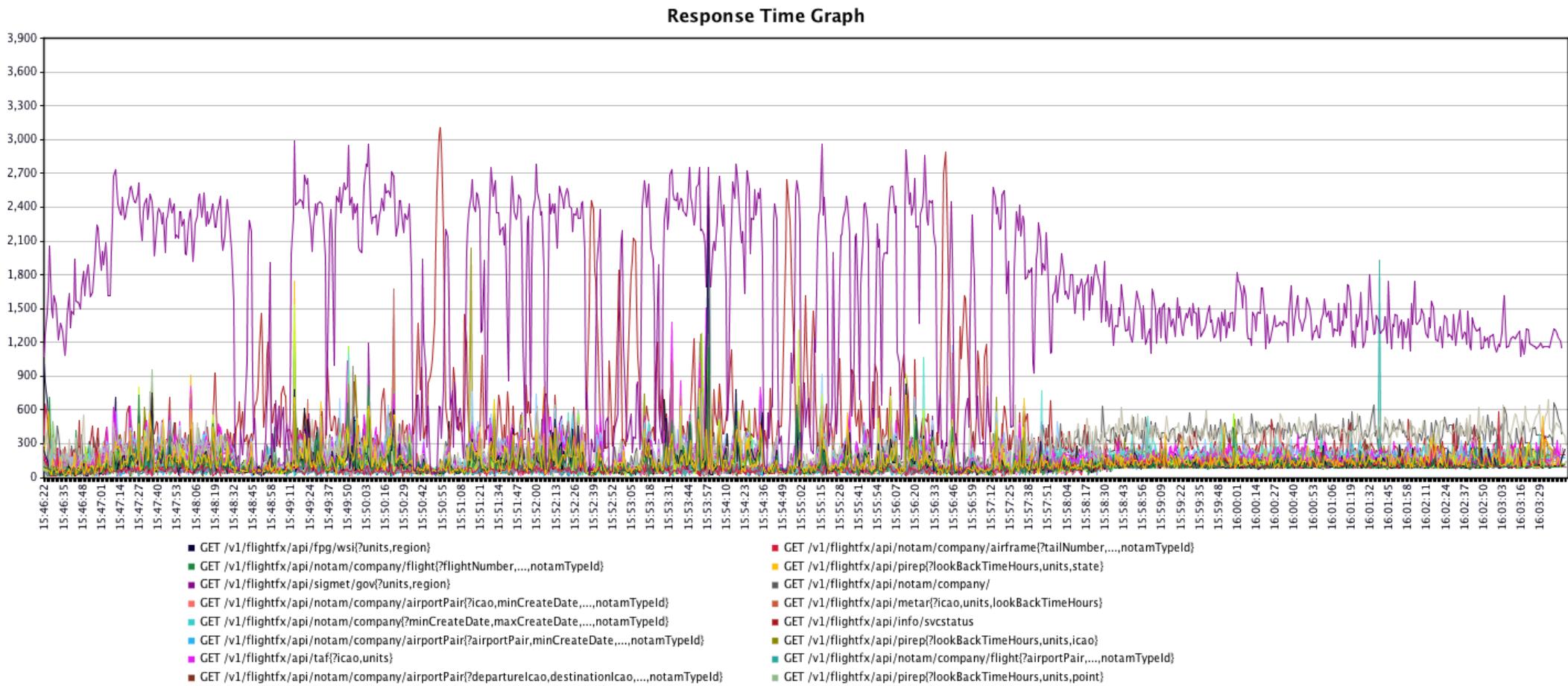
- see *report.jtl*



• open report in JMeter

Label	# Samples	Throughput	Average	Min	Max	Std. Dev.	Error %	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /v1/flightfx/api/sigmet/wsi{?units,region}	6000	5.8/sec	108	8	3277	132.96	25.08%	1.00	1.80	177.1
GET /v1/flightfx/api/airmet{?units,fir}	6000	5.8/sec	114	8	2506	136.35	24.70%	0.98	1.55	174.1
GET /v1/flightfx/api/airmet{?units,polygon}	6000	5.8/sec	153	8	1745	151.81	24.17%	175.38	1.91	31143.1
GET /v1/flightfx/api/airmet{?units,region}	6000	5.8/sec	132	8	1913	138.40	24.92%	68.60	1.56	12148.7
GET /v1/flightfx/api/fpg/wsi{?units,polygon}	6000	5.8/sec	127	8	2559	142.53	24.92%	48.52	1.92	8598.1
GET /v1/flightfx/api/fpg/wsi{?units,region}	6000	5.7/sec	124	8	3738	146.82	25.25%	0.96	1.57	171.7
GET /v1/flightfx/api/info/svcstatus	6000	5.8/sec	589	117	3406	523.56	0.00%	1.14	1.46	201.7
GET /v1/flightfx/api/metar{?icao,units,lookBackTimeHours}	6000	5.8/sec	116	8	2204	129.56	25.03%	12.27	1.84	2174.5
GET /v1/flightfx/api/mosforecast{?icao,units}	6000	5.8/sec	113	8	2515	134.56	25.38%	25.82	1.56	4587.3
GET /v1/flightfx/api/notam/company/	6000	5.8/sec	227	8	4790	194.80	25.28%	1440.35	1.54	255774.7
GET /v1/flightfx/api/notam/company/airframe{?tailNum...}	6000	5.7/sec	75	8	1041	67.72	24.78%	1.01	4.12	179.3
GET /v1/flightfx/api/notam/company/airport{?icao,minCr...	6000	5.8/sec	74	8	1064	67.61	24.52%	1.01	4.08	179.4
GET /v1/flightfx/api/notam/company/airportPair{?airport...	6000	5.8/sec	76	8	2508	85.25	25.13%	1.01	4.20	179.4
GET /v1/flightfx/api/notam/company/airportPair{?depart...	6000	5.8/sec	75	8	2437	72.71	25.07%	1.01	4.27	179.4
GET /v1/flightfx/api/notam/company/airportPair{?icao,m...	6000	5.8/sec	75	8	3017	84.30	24.52%	1.01	4.08	179.3
GET /v1/flightfx/api/notam/company/equipment{?equip...	6000	5.8/sec	75	8	1545	69.42	24.25%	1.01	4.12	179.4
GET /v1/flightfx/api/notam/company/flight{?airportPair,...}	6000	5.8/sec	77	8	7441	116.01	24.93%	1.01	4.21	179.4
GET /v1/flightfx/api/notam/company/flight{?departureIc...	6000	5.8/sec	77	8	2533	77.33	24.57%	1.01	4.25	179.4
GET /v1/flightfx/api/notam/company/flight{?flightNumbe...	6000	5.8/sec	74	8	2518	73.00	25.95%	1.01	4.11	179.2
GET /v1/flightfx/api/notam/company/flight{?icao,...,nota...	6000	5.8/sec	75	8	1408	67.22	25.38%	1.01	4.09	179.2
GET /v1/flightfx/api/notam/company/?minCreateDate,m...	6000	5.7/sec	75	8	2750	76.76	24.10%	1.01	3.76	179.5
GET /v1/flightfx/api/notam/gov{?icao,units}	6000	5.8/sec	1521	8	4145	941.37	24.37%	22.44	1.55	3982.8
GET /v1/flightfx/api/pirep{?lookBackTimeHours,units,icao}	6000	5.8/sec	174	8	7422	180.87	24.77%	0.97	1.64	173.1
GET /v1/flightfx/api/pirep{?lookBackTimeHours,units,poi...}	6000	5.8/sec	176	8	2966	149.33	23.93%	0.97	1.83	173.3
GET /v1/flightfx/api/pirep{?lookBackTimeHours,units,state}	6000	5.8/sec	139	8	3087	151.13	24.47%	8.04	1.63	1430.4
GET /v1/flightfx/api/pirep{?radius,lookBackTimeHours,u...}	6000	5.8/sec	173	8	2914	162.51	24.92%	1.57	1.73	278.9
GET /v1/flightfx/api/sigmet/gov{?units,fir}	6000	5.8/sec	115	8	1844	133.43	24.83%	1.00	1.58	177.0
GET /v1/flightfx/api/sigmet/gov{?units,polygon}	6000	5.8/sec	113	8	3102	138.15	25.68%	1.00	1.93	176.8
GET /v1/flightfx/api/sigmet/gov{?units,region}	6000	5.8/sec	117	8	3243	151.65	25.48%	0.99	1.58	177.0
GET /v1/flightfx/api/sigmet/wsi{?units,polygon}	6000	5.8/sec	110	8	7359	154.89	24.62%	6.34	2.14	1124.0
GET /v1/flightfx/api/taf{?icao,units}	6000	5.8/sec	206	8	1723	159.42	24.08%	4.15	1.51	738.6
GET /v1/flightfx/api/vaaccloud{?units}	6000	5.8/sec	115	8	3418	139.88	25.08%	32.04	1.50	5674.8
GET /v1/flightfx/api/vaacvolcano{?units}	6000	5.8/sec	110	8	3307	134.70	24.48%	38.13	1.51	6755.8
GET /v1/flightfx/api/windsaloft{?icao,units}	6000	5.8/sec	216	8	3091	160.17	24.28%	10.46	1.55	1857.7
POST /v1/flightfx/api/notam/company	6000	5.8/sec	232	8	2402	189.73	24.83%	1445.50	8.59	257315.0
TOTAL	210000	200.7/sec	176	8	7441	333.14	24.11%	3344.77	91.79	17061.7

- open report in JMeter



- destroy infrastructure after test is done:
terraform plan -destroy -out plan.output

```
- module.ohio.aws_vpc.qa_load_test_vpc  
  
- module.oregon.aws_instance.server  
  
- module.oregon.aws_internet_gateway.qa_load_test_internet_gateway  
  
- module.oregon.aws_key_pair.qa_load_keypair  
  
- module.oregon.aws_route_table.qa_load_test_route_table  
  
- module.oregon.aws_route_table_association.qa_load_test_route_table_association  
  
- module.oregon.aws_security_group.qa_load_test  
  
- module.oregon.aws_subnet.qa_load_test_subnet  
  
- module.oregon.aws_vpc.qa_load_test_vpc
```

Plan: 0 to add, 0 to change, 40 to destroy.

This plan was saved to: plan.output

- apply destroy plan:

terraform apply plan.output

```
module.ireland.aws_internet_gateway.qa_load_test_internet_gateway: Still destroying... (ID: igw-ad39)
module.ireland.aws_internet_gateway.qa_load_test_internet_gateway: Destruction complete after 11s
module.ireland.aws_security_group.qa_load_test: Still destroying... (ID: sg-93fcb1ee, 4m11s elapsed)
module.ireland.aws_security_group.qa_load_test: Still destroying... (ID: sg-93fcb1ee, 4m21s elapsed)
module.ireland.aws_security_group.qa_load_test: Still destroying... (ID: sg-93fcb1ee, 4m31s elapsed)
module.ireland.aws_security_group.qa_load_test: Still destroying... (ID: sg-93fcb1ee, 4m41s elapsed)
module.ireland.aws_security_group.qa_load_test: Still destroying... (ID: sg-93fcb1ee, 4m51s elapsed)
module.ireland.aws_security_group.qa_load_test: Still destroying... (ID: sg-93fcb1ee, 5m1s elapsed)
module.ireland.aws_security_group.qa_load_test: Destruction complete after 5m10s
module.ireland.aws_vpc.qa_load_test_vpc: Destroying... (ID: vpc-3966185f)
module.ireland.aws_vpc.qa_load_test_vpc: Destruction complete after 2s
```

```
Apply complete! Resources: 0 added, 0 changed, 40 destroyed.
```

Special thanks to



Yuriy Maftiyak

ITea&Coffee

See you on our next
meeting in September!

06 / 09 / 2018