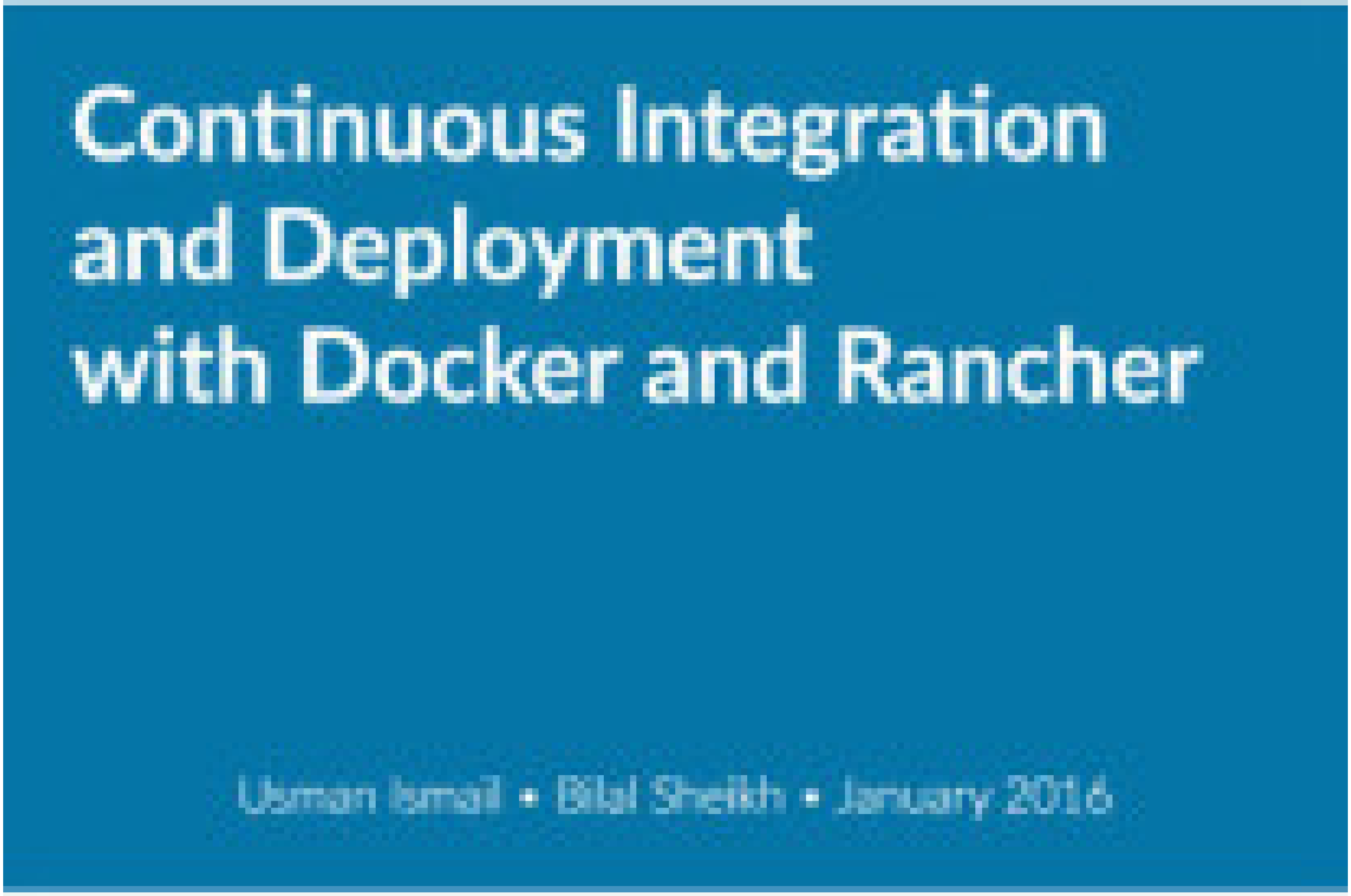
Docker based build pipelines (Part 3) - Managing Production Environments

| FEBRUARY 1, 2016



Continuous Integration and Deployment with Docker and Rancher

Usman Ismail • Bilal Sheikh • January 2016

· in this series of articles we have looked at creating continuous integration pipelines using Jenkins and continuously deployingto
·ation environments. We also looked at using Rancher compose to run deployments as well as Route53 integration to do basic DN
gement. Today we will cover production deployments strategies and also circle back to DNS management to cover how we can ru
region and/or multi-data-center deployments with automatic fail-over. We also look at some rudimentary auto-scaling so that we
·atically respond to request surges and scale back when request rate drops again. If you'd like to read this entire series, we've ma
· \"Continuous Integration and Deployment with Docker and Rancher\" available for download.

eployment Strategies

·f challenges when managing production environments is to ensure minimal or zero downtime during releases. Doing so predictab
afely, takes quite a bit of work. Automation and quality assurance can go a long way to make releases more predictable and safe.
failures can and do happen, and for any good ops team the goal would be to recover quickly while minimizing impact. In this secti
·over a few strategies for running production deployments and their trade-offs for updating your application.

-place updates

rst strategy is called *in-place update*, as the name suggests, the idea is to re-use the production environment and update the
·ation in-place. These are also sometimes referred to as *Rolling Deployments.* We're going to work with our sample application (g
we covered in part 1 and part 2. Further, we're going to assume that you have the service running with Rancher. To do an in-place
e, you can use the upgrade command:

```
:h_version=${GO_AUTH_VERSION} rancher-compose --project-name go-auth \
-url http://YOUR_RANCHER_SERVER:PORT/v1/    \
-access-key <API_KEY>                       \
-secret-key <SECRET_KEY>                    \
· verbose up -d --force-upgrade --pull auth-service
```

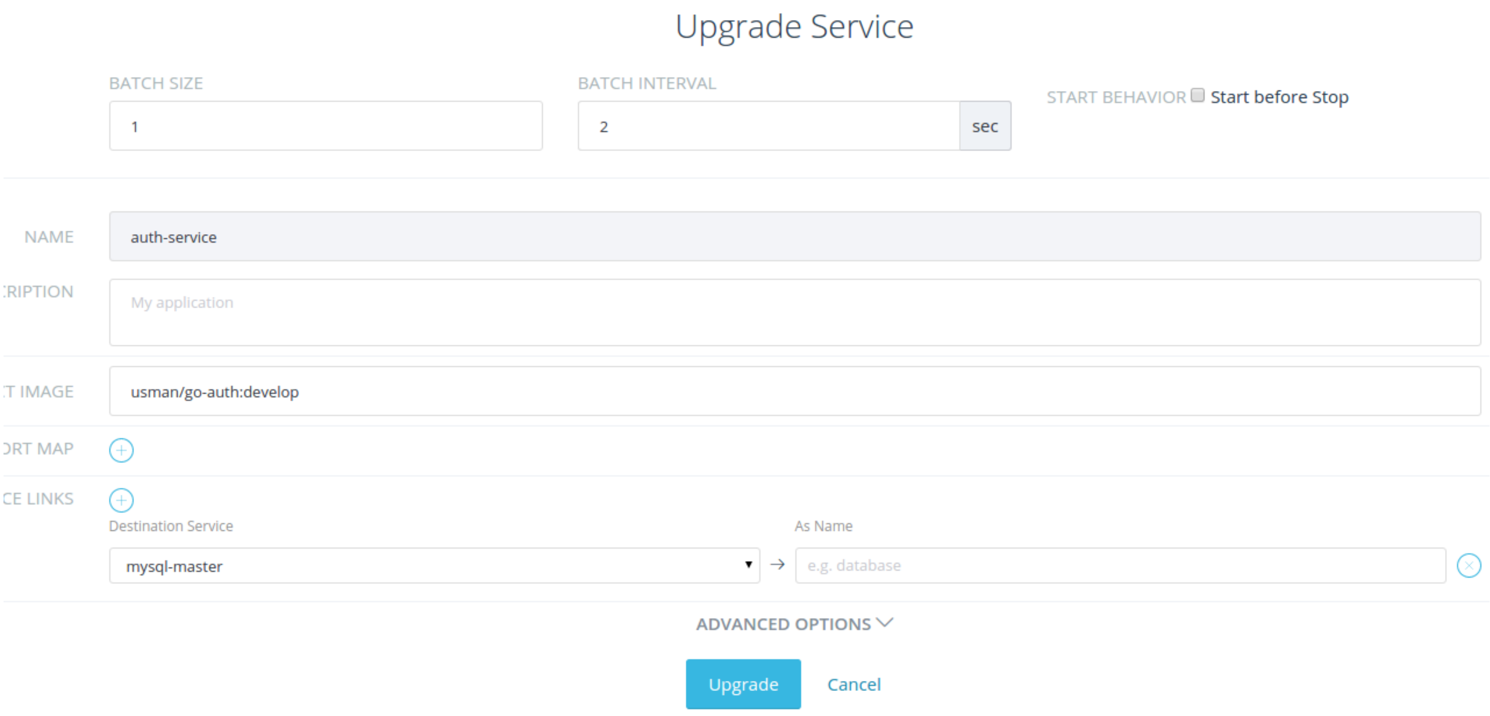
d the scenes, Rancher agent fetches the new image on each host running an auth-service container. It then stops the old containe ıunches new containers in batches. You can control the size of the batch by using the *--batch* flag. Additionally, you can specify a ɜ interval *(--interval)* between batch updates. A large enough interval can be used to allow you to verify that the new containers ar ɩing as expected, and on the whole, the service is healthy. By default, old containers are terminated and new ones are launched in . Alternatively, you can tell Rancher to start the new containers before stopping the old containers by setting the *start_first* flag in er-compose.yml.

```
:h-service:
ıpgrade_strategy:
  start_first: true
```

are not happy with the update and want to roll-back, you can do so with the rollback flag for the upgrade command. Alternatively, ɩant to proceed with the update, simply tell Rancher to complete the update by specifying the confirm-update flag.

```
:h_version=${GO_AUTH_VERSION} rancher-compose --project-name go-auth \
-url http://YOUR_RANCHER_SERVER:PORT/v1/    \
-access-key <API_KEY>                       \
-secret-key <SECRET_KEY>                    \
· verbose up -d --[rollback|confirm-upgrade] auth-service // to confirm or rollback an upda
```

an also perform these updates using the Rancher UI, by selecting \“upgrade\” from a service’s menu (shown below).



Upgrade Service

BATCH SIZE: 1

BATCH INTERVAL: 2 sec

START BEHAVIOR □ Start before Stop

NAME: auth-service

:RIPTION: My application

:T IMAGE: usman/go-auth:develop

ЭRT MAP

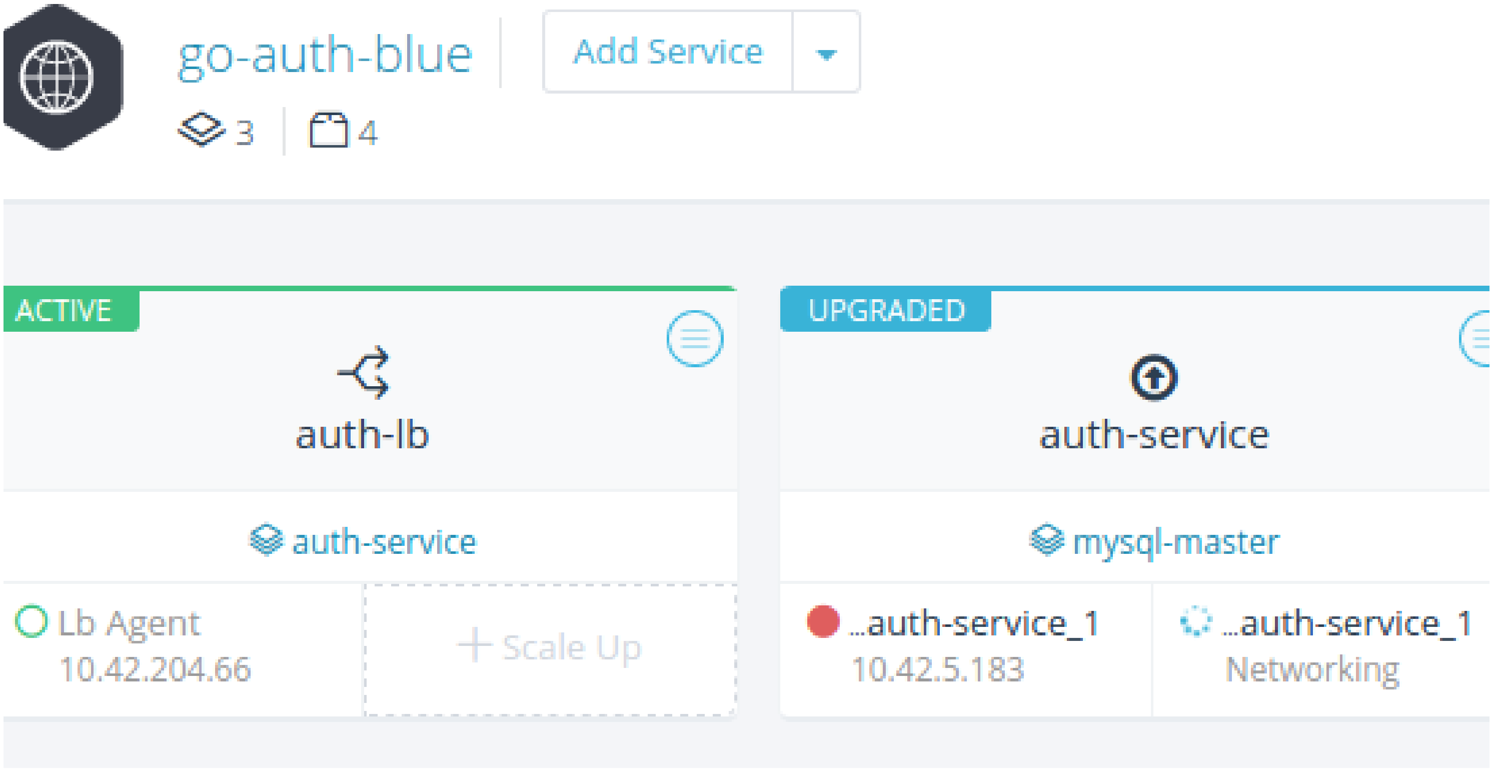
CE LINKS

Destination Service: mysql-master → As Name: e.g. database

ce updates are quite simple to perform and don’t require the additional investment to manage multiple stacks. There are, howeveı sides to this approach for production environments. First, it is typically difficult to have fine-grained control over rolling updates, i. end to be unpredictable under failure scenarios. For example, dealing with partial failures and rolling back a rolling update can ge messy. You have to know which nodes were deployed too, which failed to deploy and which are still running the previous revision. ıd, you have to make sure all updates are not only backwards compatible but also forward compatible because you will have old a ɪersions of your application running concurrently in the same environment. Last, depending on the use case, in-place updates migl ıctical. For example, if legacy clients need to continue to use the old environment while newer clients roll forward. In this case ating client requests is much easier with some of the other approaches we are going to list today.

ue-Green Deployments

of predictability is a common problem with in-place updates. To overcome that, another strategy for deployments is to work with t
el stacks for an application. One active and the other in standby. To run a new release, the latest version of the application is depl
standby stack. Once the new version is verified to be working, a cut-over is done to switch traffic from the active stack to the star
. At that point the previously active stack becomes the standby and vice versa. This strategy allows for verification of deployed cc
ollbacks (switching standby vs active again) and also extended concurrent operation of both stacks if needed. This strategy is
ıonly referred to as blue-green deployments. To accomplish such deployments through Rancher for our sample application, we ca
y create two stacks in Rancher: go-auth-blue and go-auth-green. Further, we're going to assume that the database is not part of th
s and is being managed independently. Each of these stacks would just run *auth-service* and *auth-lb* services. Assuming that go-a
stack is live, to perform an update, all we need to do is to deploy the latest version to the blue stack, perform validation and switc
over to it.



ffic Switch

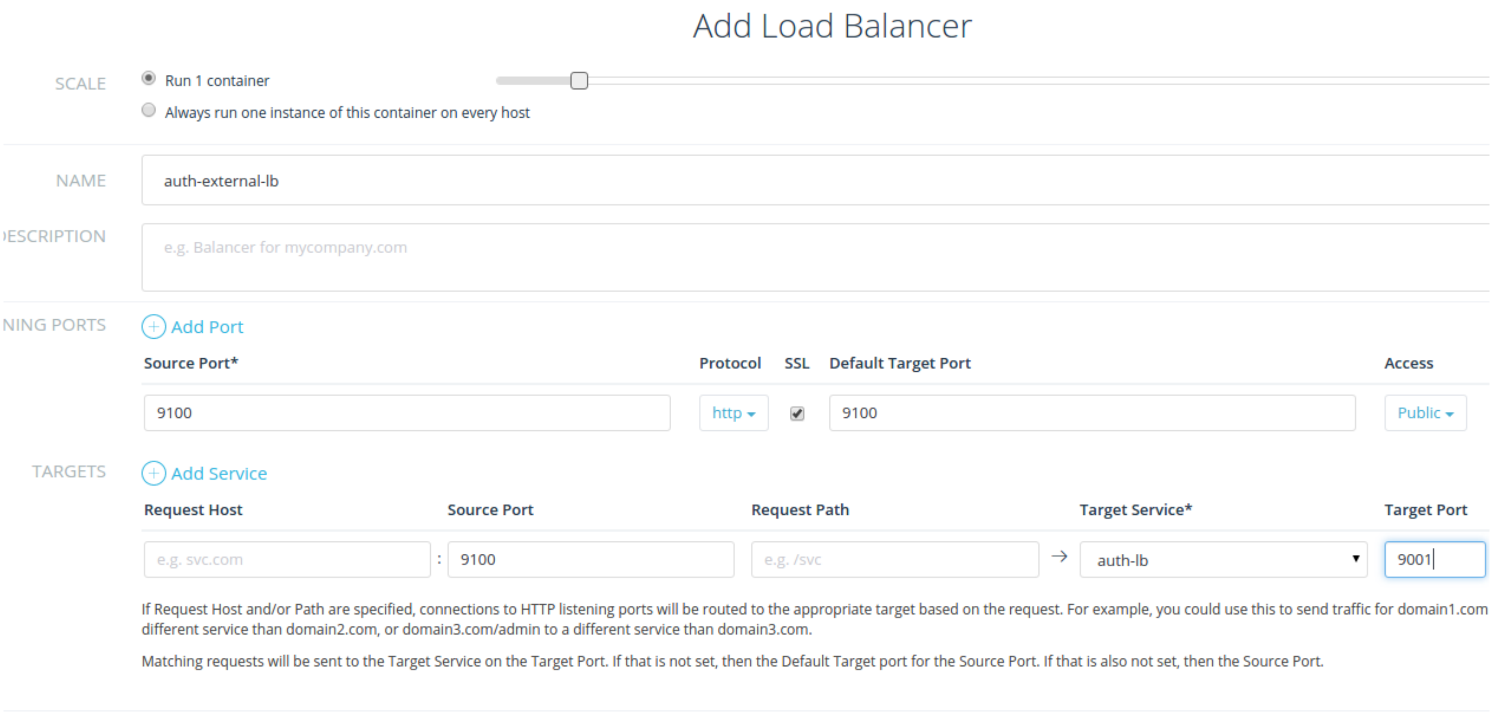
are a couple of options for doing the traffic switch, changing DNS records to point to the new stack or using a proxy or load-balar
outing traffic to the active stack. We cover both options in detail below.

S record update

ple approach is to update a DNS record to point to the active stack. One advantage of this approach is that we can use weighted I
ds (Details later) to slowly transition the traffic over to the new version. This is also a simple way to do canary releases which are
l for safely phasing in new updates on live environments or for doing A/B tests. For example, we can deploy an experimental featu
vn feature stack (or to the in-active stack) and then update the DNS to forward only a small fraction of the traffic to the new versio
is an issue with the new update, we can reverse the DNS record changes to roll back. Further, it is much safer than doing a cut-ov
all traffic switches from one stack to another, which can potentially overwhelm the new stack. Although simple, DNS record upda
e cleanest approach if you want all your traffic to switch over to the new version at once. Depending on the DNS clients, the chang
ake a long time to propagate, resulting in a long tail of traffic against your old version instead of a clean switch over to the new ver

ng a reverse proxy

a proxy or load-balancer and simply updating it to point to the new stack is a cleaner way of switching over the entire traffic at-or ıpproach can be quite useful in various scenarios, e.g., non-backwards compatible updates. To do this with Rancher, we first need reate another stack which contains a load-balancer only. [](http://cdn.rancher.com/wp-content/uploads/2015/11/29204307/add-l ıal.png)



Add Load Balancer

SCALE ● Run 1 container
○ Always run one instance of this container on every host

NAME auth-external-lb

ESCRIPTION e.g. Balancer for mycompany.com

NING PORTS ⊕ Add Port

Source Port*	Protocol	SSL	Default Target Port	Access
9100	http	☑	9100	Public

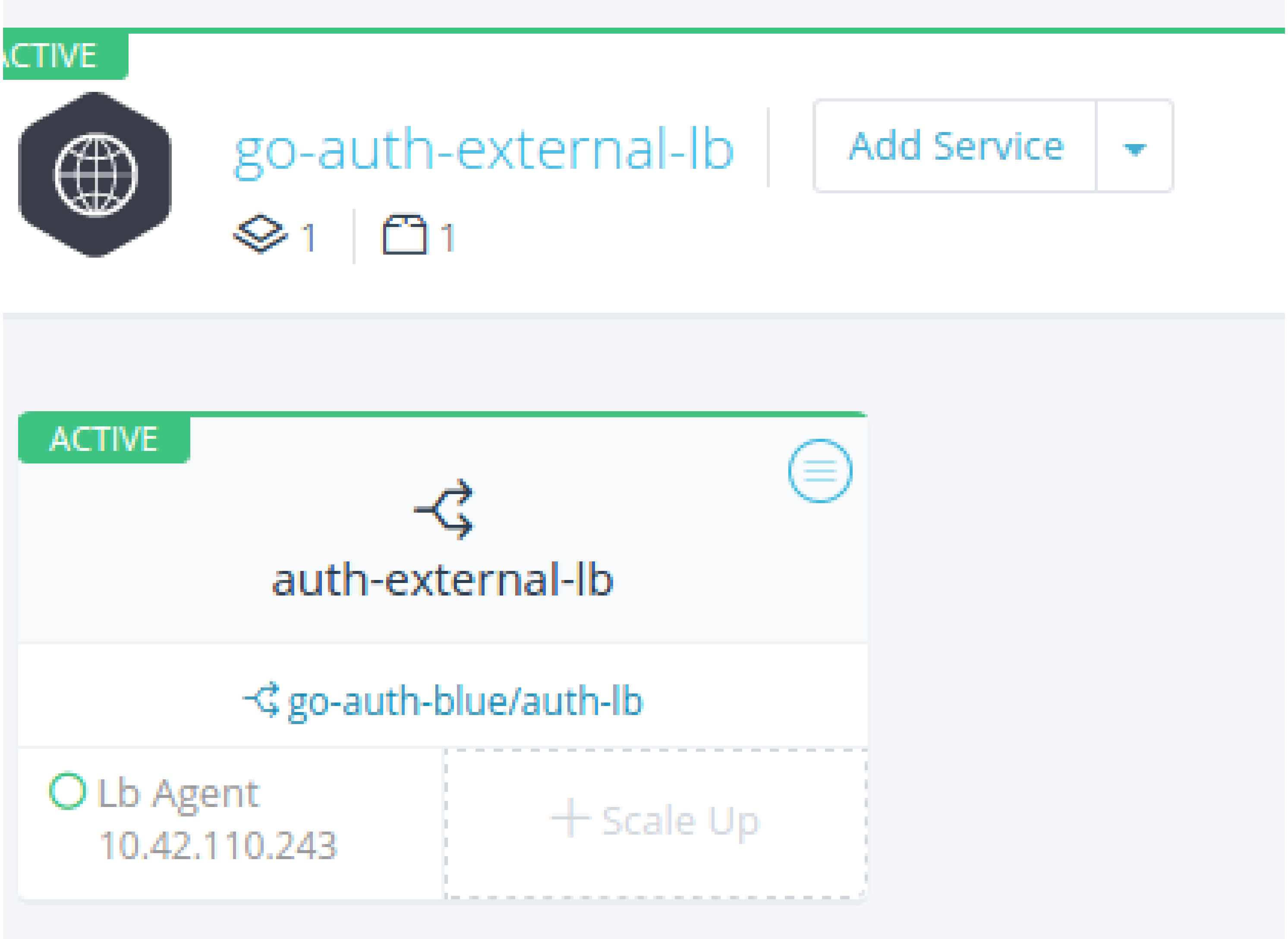
TARGETS ⊕ Add Service

Request Host	Source Port	Request Path	Target Service*	Target Port
e.g. svc.com	9100	e.g. /svc	auth-lb	9001

If Request Host and/or Path are specified, connections to HTTP listening ports will be routed to the appropriate target based on the request. For example, you could use this to send traffic for domain1.com different service than domain2.com, or domain3.com/admin to a different service than domain3.com.

Matching requests will be sent to the Target Service on the Target Port. If that is not set, then the Default Target port for the Source Port. If that is also not set, then the Source Port.

we specify a port for the load-balancer, configure SSL and pick the load-balancer for the active stack as the *target service* from th down menu to create the load-balancer. Essentially we are load balancing to a load-balancer which is then routing traffic to actual e nodes. With the external load-balancer, you don't need to update the DNS records for each release. Instead, you can simply upda xternal load-balancer to point to the updated stack.

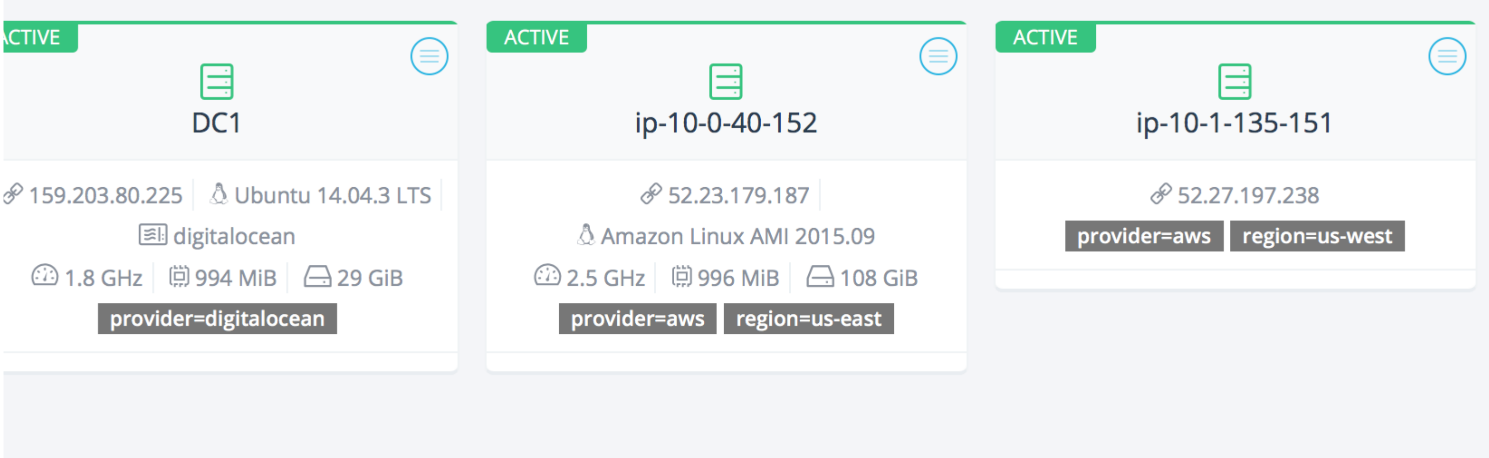


ulti-Region, Multi-Cloud deployments

:hat we are deloying production deployments we need to consider availability. For example Amazon's SLA supports a 99.95% up ti ich region without incurring penalties for Amazon. This is in the three nines availability bracket and is normally considered a minir ost large scale customer services. For more critical services 5 nines of up time are a more acceptable target. To get there we will ce resources in multiple amazon availability zones. We can also use of Rancher to manage cross-cloud provider redundancy howe evel of redundancy is not required in most deployments.

unching Tagged Instances

rst step in making multi-region or cloud deployments is to launch Rancher Compute nodes in multiple AWS regions. You may also h compute nodes on other cloud providers such as Digital Ocean. Use labels to tag each host with the provider and region. The fig shows a three node cluster with one node in AWS US East, one in US West and one in Digital Ocean.


CTIVE
DC1
159.203.80.225 | Ubuntu 14.04.3 LTS
digitalocean
1.8 GHz | 994 MiB | 29 GiB
provider=digitalocean

ACTIVE
ip-10-0-40-152
52.23.179.187
Amazon Linux AMI 2015.09
2.5 GHz | 996 MiB | 108 GiB
provider=aws region=us-east

ACTIVE
ip-10-1-135-151
52.27.197.238
provider=aws region=us-west

oss-Region/Cloud Security

you have to configure your Rancher server security group to allow access from the compute nodes in remote regions. This needs
using CIDR blocks as security group based white lists do not work across regions. If you would like a more secure solution you wi
to use VPC Peering Connections and Direct Connect or VPNs to connect across regions. However, for the purposes of this article
sing very permissive security group rules and relying on Rancher security instead.

date Compose Templates

hat we have compute nodes in two amazon regions and also in Digital Ocean we will update our docker-compose.yml and ranche
ose.yml to launch services in the various regions. In the docker-compose.yml find the auth-service tag and copy its content two m
and name the three tags as follows. Similarly find the auth-lb tag and copy it two more times and rename to reflect region and
ler.

```
h-service-aws-east:
ty: true
ommand:
 ...
h-service-aws-west:
ty: true
ommand:
 ...
h-service-digitalocean:
ty: true
ommand:
 ...
```

lition add the *io.rancher.scheduler.global *label to all three auth-service definitions as well as the three load-balancer definitions.
nsure that there is an instance of the container running on all hosts subject to the filter defined in *io.rancher.scheduler.affinity*. In t
y label define where you want the service or load-balancer instances to be run. For example the entry below shows the affinity for
iners running in aws us-west. With this setup we ensure that we have at least one auth service instance and one load-balancer
nce in each of our two aws regions and one in digital ocean.

```
els:
  io.rancher.scheduler.global: 'true'
  io.rancher.scheduler.affinity:host_label: provider=aws,region=us-west
```

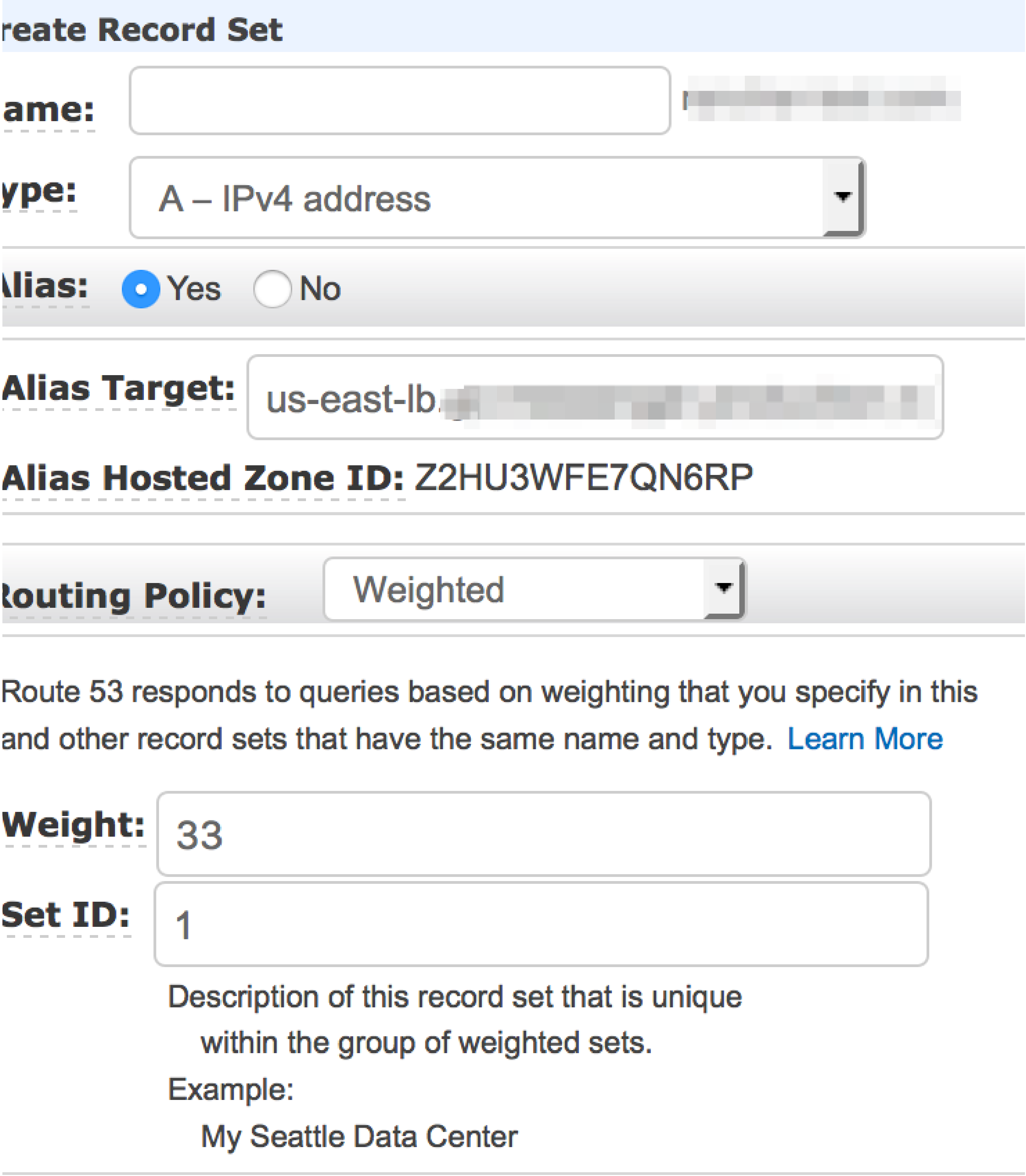
up DNS

:hat we have all our containers defined we can start setting up DNS to route traffic and fail over automatically. To start off, follow t ctions here to setup route53 integration. Note that we will get three separate records, one for each of the load-balancers services record may have multiple IPs depending on how many containers are running in each service. Having DNS entries to aggregate th iners in each region is useful but to be truly cross-region you must present and single domain to external clients and have the serv oute traffic efficiently. We have several ways of setting up this routing in Route53 namely: Weighted, Geo-location, and Latency ba: are pros and cons to each approach which we will discuss next.

ighted Routing

nted routing allows you to specify the portion of traffic that goes to each region and fail-over automatically if one of the regions gc . This is a good way to control how much traffic goes to each region/cloud. For example we may want to keep the majority of our t VS servers as they tend to be more performant. Note this strategy can also be used for traffic shifting during blue-green deployme ercentage of traffic going to each of the load-balancers will be more or less stable. The downside is that this strategy does not tal ccount where your traffic originates. The configured percentage of traffic will go to us-west regardless of whether the source of th : is New York or Seattle.

e Weighted routing policy browse to AWS Console > Route 53 > Hosted Zonesand select your hosted zone. Now click *Create Reco* ı the screen on the right select a name for your sub-domain (e.g. go-auth-prod) and select type A as Rancher creates A type record load-balancer. Now select yes on the Alias and select the name of one of the auto-created DNS entries as the alias target. For exa ıve selected the us-east load-balancer as our target. Select *Weighted* as the Routing policy and give this this route a weight and se he set id. Repeat the same process for the other two load-balancers making sure to use same Name, but unique Set ID for all thre d sets. The traffic will now be split based on the relative weight of each route. If you used a weight of 33 for each of the routes the regions will get a third of the weight.



reate Record Set

ame: [redacted]

ype: A – IPv4 address

lias: ● Yes ○ No

Alias Target: us-east-lb.[redacted]

Alias Hosted Zone ID: Z2HU3WFE7QN6RP

outing Policy: Weighted

Route 53 responds to queries based on weighting that you specify in this and other record sets that have the same name and type. Learn More

Weight: 33

Set ID: 1

Description of this record set that is unique within the group of weighted sets.

Example:

My Seattle Data Center

ency Based Routing

atency based routing traffic will be sent to the AWS data-center with the lowest latency from the client sending the DNS query. Thi ss the draw back of Weighted routing, i.e. clients frm New York will be sent to US East where as clients from Seattle are sent to U: The drawback of this approach is that you cannot balance traffic across regions. For example if 90% of your users are in New Yor

›f your traffic will go the US East and your US West deployment will sit mostly idle. This may be fine for most use-cases but may be table if you want to keep deployments of roughly equal size. Also peak utilization is strongly correlated to time of day the traffic o region will be more variable through the day because of time zone differences.

tup Latency based routing follow the exact same procedure as weighted but select Latency as the Routing Policy. In addition you e asked to specify a Amazon Regions instead of a weight for each of the three route entries. For US-East and US-West select the ctive regions, for Digital Ocean select the AWS region closest to your DigitalOcean region. For example if you launched nodes in lOcean's NYC datacenter then you should select US-East as the region for latency based routing to digital ocean. Traffic from the will be split evenly between your load-balancer in AWS US East and DigitalOcean.

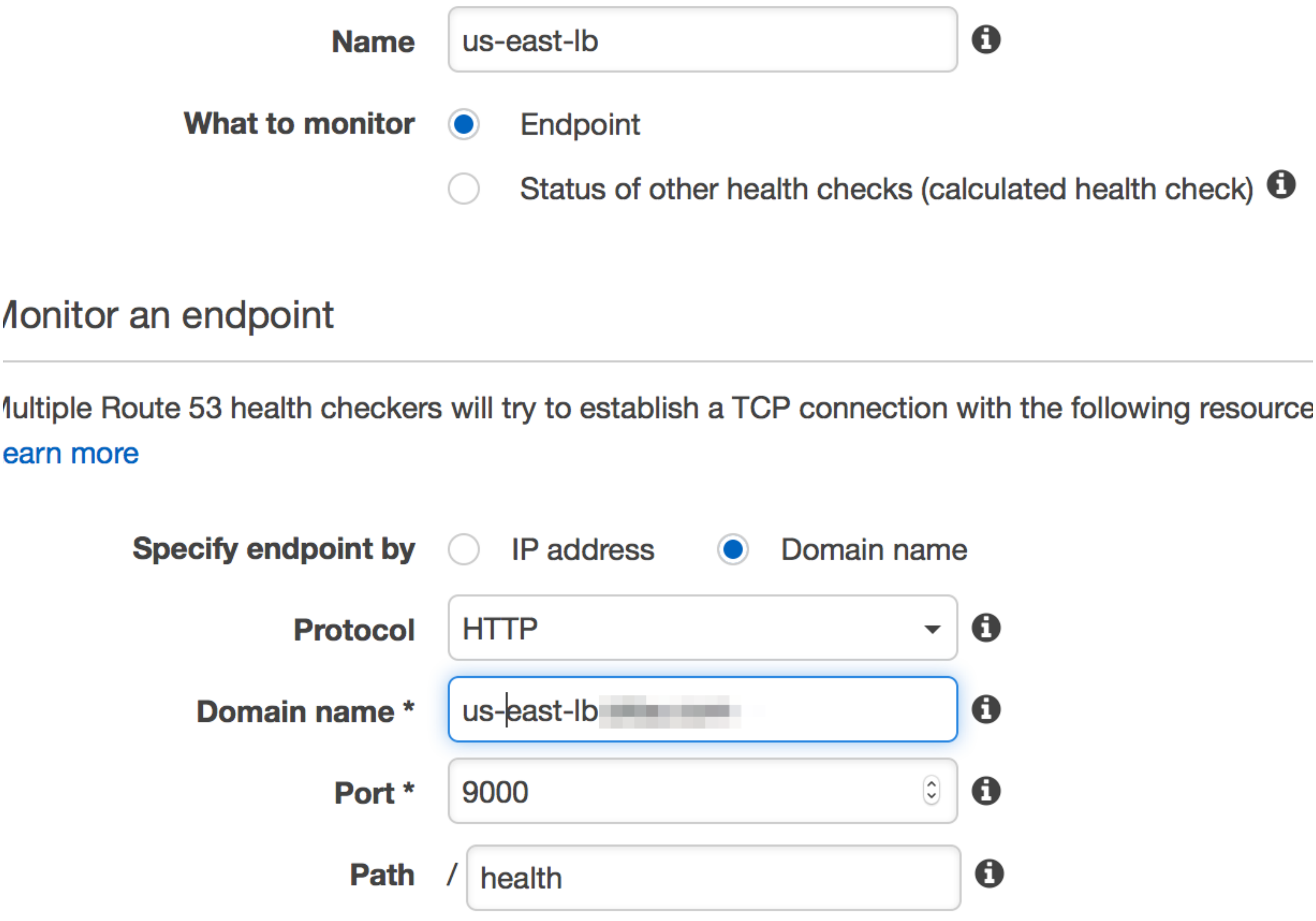
ɔ-location Based Routing

, you can use geo-location based routing to explicitly specify where traffic originating in each region must go. In practice this is ve r to Latency based routing however, you can explicitly set region to target matching rather than relying on latency. This allows you Portugal and Brazil to the same target (You have your Portuguese localized servers in that target). The down side of this approach ay increase latency by routing to a far away cluster. And the granularity is Continent and Country only, and some countries are ver

eo-location based routing follow a similar process of creating route sets but select *Geolocation* as the routing policy. In this case ave to add one record set for each country or continent for which you would like to route traffic to a specific data center. You shou specify a record set for the *default* location in order to avoid having to explicitly specify entries for all possible countries and regio

S Health Check

would like auto-failover for your various routes (regardless of routing policy) then you have to create DNS health checks. To do so 'S Console > Route 53 > Health Checksand click *Create Health check.* In the configure health check screen enter a name for your h , select *Endpoint* monitoring, and Domain name as endpoint. The protocol should be http (or https if you set it up), the domain na d be one of the ones created by Rancher that we used as Alias targets earlier. Specify the port as 9000 and the path as health. (No he go-auth service has a health check end point at /health). You should create a health check for each of the three load-balancers ed in Rancher.



Name: us-east-lb

What to monitor: ● Endpoint ○ Status of other health checks (calculated health check)

/lonitor an endpoint

/lultiple Route 53 health checkers will try to establish a TCP connection with the following resource
earn more

Specify endpoint by: ○ IP address ● Domain name

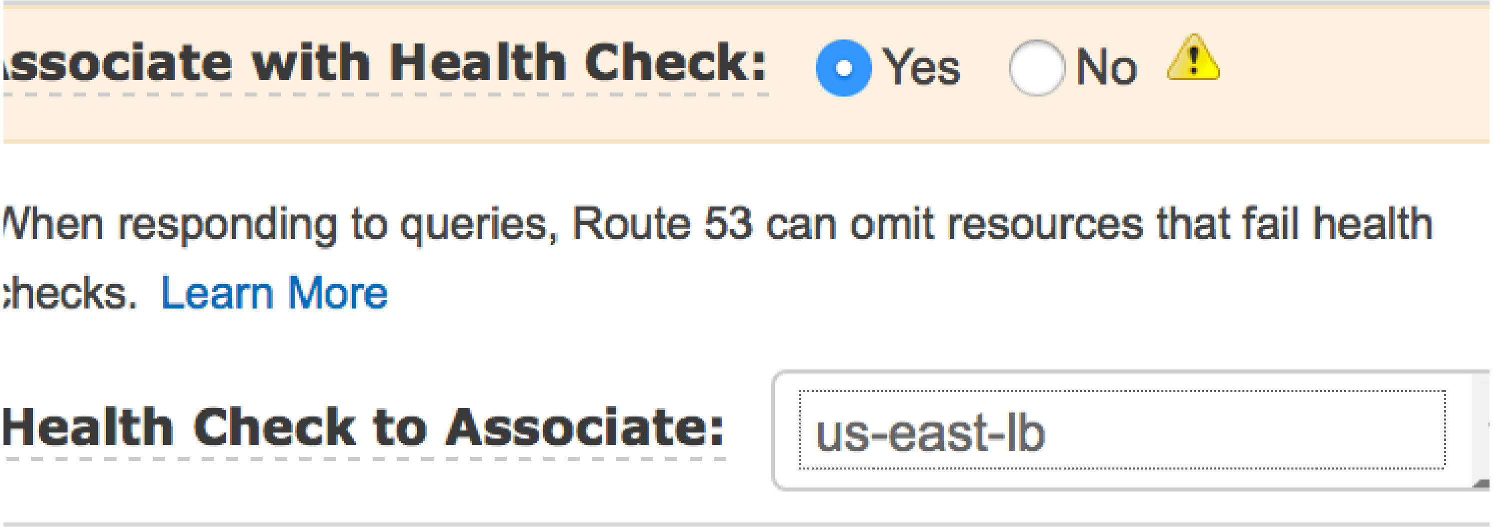
Protocol: HTTP

Domain name *: us-east-lb[redacted]

Port *: 9000

Path: / health

that you have the health checks created, go back to your hosted zone. For each of the aggregate record sets you created (i.e. the the weighted/latency/geolocation routing) select yes for the *Associate with Health Check* setting and select the relevant health c vill mean then if the health checks for all containers in a given region fails, route53 will automatically take that route out of the rot utomatically switch all traffic to the remaining routes. This will help you react to outages in any one region or cloud provider witho uman intervention. Such automatic fail-over for high availability services.]



ssociate with Health Check: ● Yes ○ No ⚠

Vhen responding to queries, Route 53 can omit resources that fail health :hecks. Learn More

Health Check to Associate: us-east-lb

uilding Auto-scaling Arrays

f the significant difference between a production environment and the testing environments we covered earlier is that the load is ole and unpredictable. One of the major benefits of cloud-based and container based deployments is to minimize the overhead, bo cial and technical, of dealing with this variability. Auto scaling arrays are an important part of realizing these goals. They can help ffic spikes without human intervention and also help you save money by scaling resources up and down as you move through your traffic peak and troughs. We will use Amazon's Auto-scaling arrays with Rancher to scale our service stacks.

eating a launch configuration

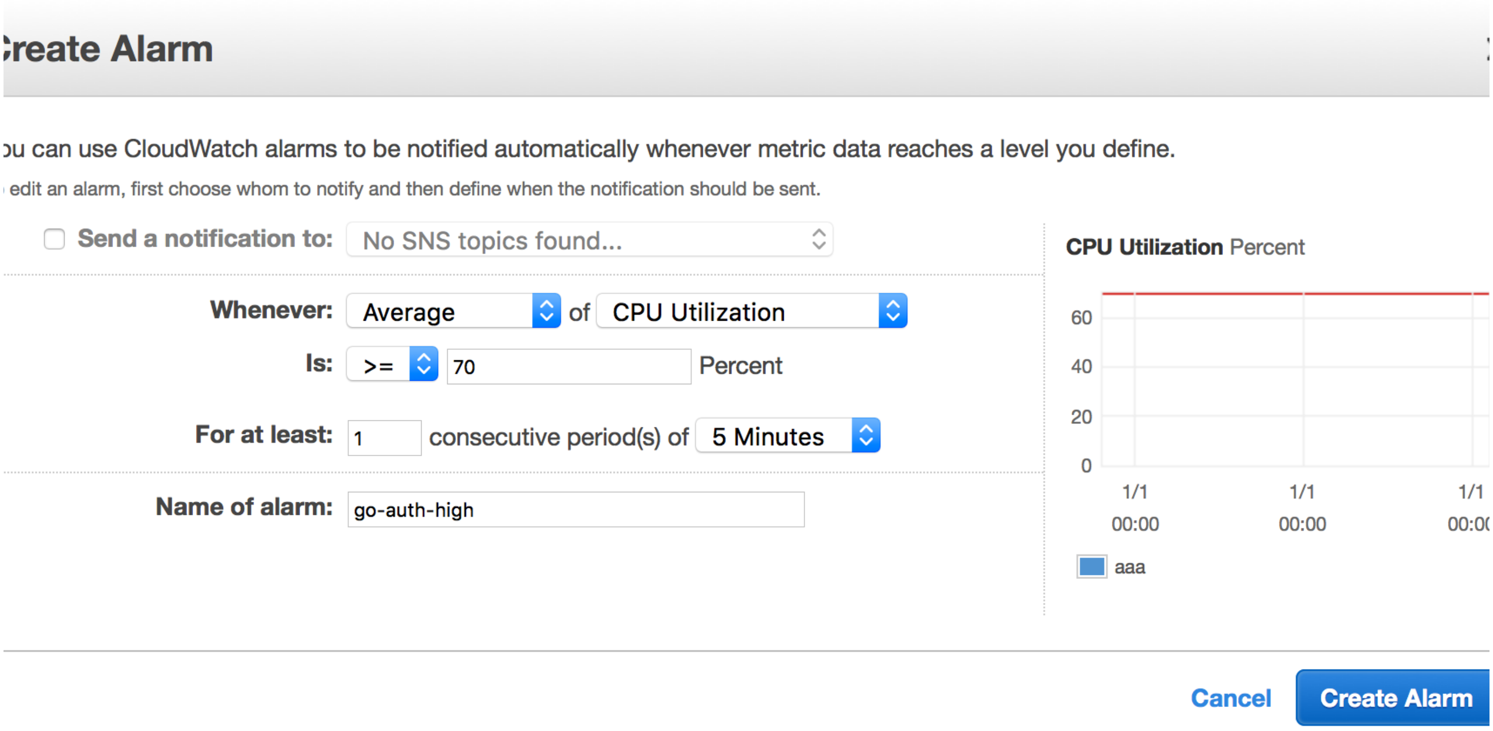
rst step in creating your auto scaling arrays is to create a launch configuration. To do so, go to *Amazon Console > Ec2 > Launch guration* and select *Create Launch Configuration.* Follow the screen instructions to create a launch configuration using an AMI of y e (we normally Amazon's stock Linux AMI). When you get to step 3 *Configure Details*, select *Advanced Details* and user data secti the commands shown below. In this we are using cloud-init to install docker and run our compute instance. Note that we tag our ute node instance with the name of the service. The Rancher URL needed for docker run can be retrieved from */[RANCHER_SERVER]:[RANCHER_PORT]/infra/hosts/add/custom.* It is specific to your server and environment.

```
/bin/bash

Update all packages to pull in latest security updates etc
 update -y

Install docker
 install docker -y
rvice docker restart

Start Rancher Compute node
ker run
  -e CATTLE_HOST_LABELS='service=[SERVICE_NAME]'  \
  -d --privileged                                 \
  -v /var/run/docker.sock:/var/run/docker.sock    \
  rancher/agent:v0.8.2                            \
  http://[RANCHER_SERVER]:[RANCHER_PORT]/v1/scripts/EFA4EAD....
```

eating Auto-scaling Group

hat you have your launch configuration you can create a new auto scaling array. To do so browse to *AWS Console > Ec2 > Autosc s* and select *Create Auto scaling Group.* On the first screen select the launch configuration that you created in the previous sectio o select *Use scaling policies to adjust the capacity of this group* and specify the scaling range in the scale between section. Note bly best practice to estimate your maximum need and double it for the maximum scale option.

Increase Group Size section select *Add new alarm.* In the Create Alarm screen uncheck the send notification box and specify a sc e. For example we will scale up when the average CPU utilization of the array is higher than 70% for five minutes. Similarly select *ew alarm* option in the of the *Decrease group size* section and specify and alarm for when average CPU utilization is lower then 1 e minutes.



reate Alarm

ou can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.

edit an alarm, first choose whom to notify and then define when the notification should be sent.

□ Send a notification to: No SNS topics found...

Whenever: Average of CPU Utilization

Is: >= 70 Percent

For at least: 1 consecutive period(s) of 5 Minutes

Name of alarm: go-auth-high

Take Action field of Increase Group size section choose *Add 1 instance* and similarly in the Decrease Group Size section choose *ve 1 instance*. Follow the remaining steps to create your auto scaling array and your auto scaling array is ready to go. We have us based alarm however, you can use any cloudwatch metric you like to create alarms (and therefore scaling policies).

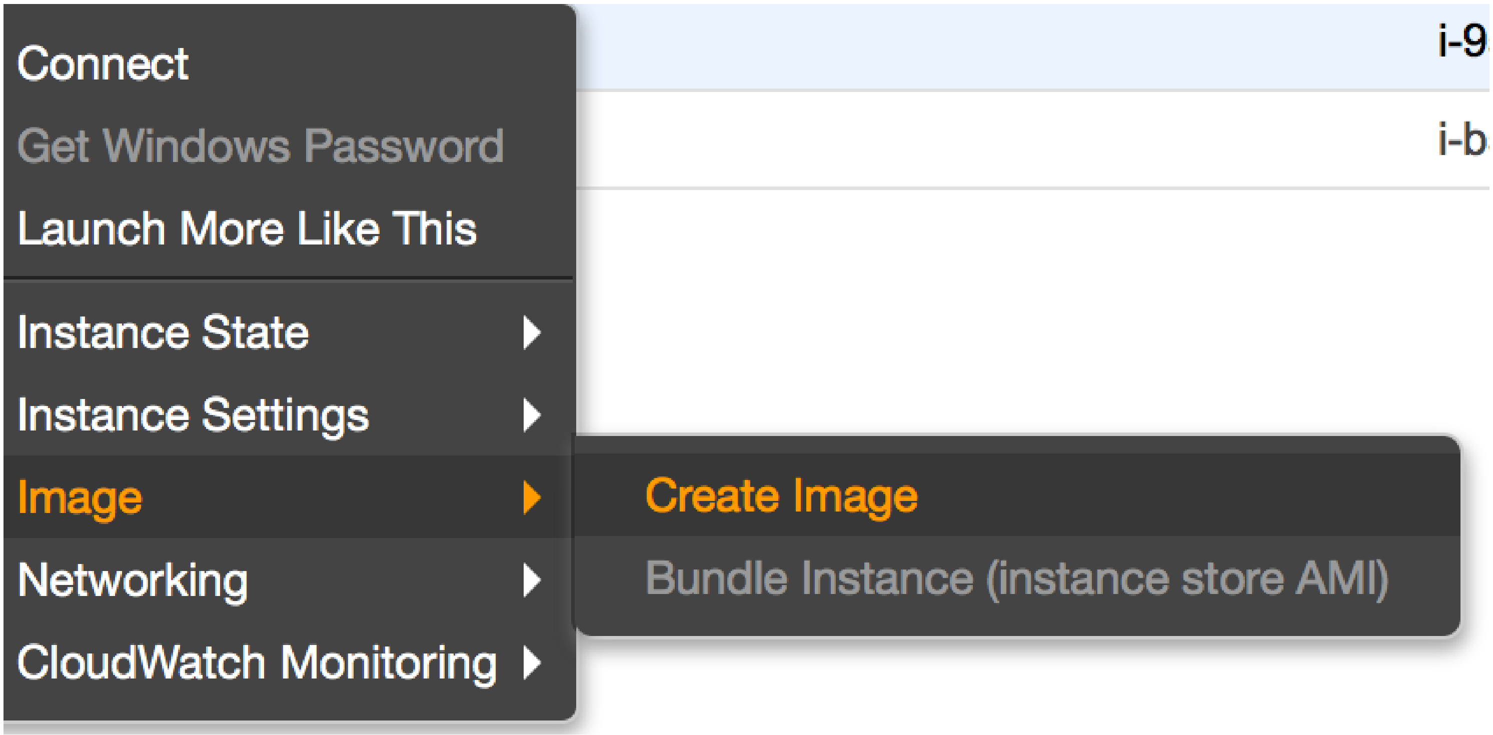
s point our auto scaling array is ready to scale out to support additional containers when CPU utilization gets high. When new nces come up they register themselves with Rancher and are available for container launches however launching container on the is still a manual process. We can automate this using a small modification to the docker-compose.yaml we defined for our servic rlier article. We need to add the following two labels to the aut-service entry. This will specify that we want to enable global sched ntainers fort this service and that we want to launch a container of this service on every host with the label *service* with value equ ervice name that we specified earlier in our launch configuration in the *CATTLE_HOST_LABELS*.

```
h-service:
abels:
 io.rancher.scheduler.global: 'true'
 io.rancher.scheduler.affinity:host_label: service=[SERVICE_NAME]
```

his change every new autoscaled host will now automatically run an instance of your service container. This will allow you to scal laily traffic peak and release resources when you scale back down all without any human intervention. A feature soon to be releas ner allows you to run multiple service instances on each of the autoscaled hosts and thus better utilize host resources. This will h ther reduce the cost of running our servers as well as give us more redundancy in case of container failures.

e Custom AMI

ptimization that you can apply when using auto-scaling arrays is to use a custom AMI with docker in which you apply all the lates ity updates, install docker and download the required Rancher client container image. This will help you shave precious seconds c h time if you need to scale out a large number of instances quickly. In addition this will make your scaling out operations indepen ckerHub and Package Manager (yum/apt) repositories. This could be critical if for example you need to launch instances while erhub is down for maintenance. The easiest way to do this is to launch an Instance using an AMI of your choice, ssh into the insta pply the required commands. Once you have done so you can right-click the instance in the Amazon Console and select *Create Im* an now use the image as your launch configuration AMI.



example, if you used the Amazon linux AMI (ami-60b6c60a in us east region), then you can use the following commands to get yo
eady.

```
 update -y
 install docker -y
ker restart
ker run -d --privileged                              \
  -v /var/run/docker.sock:/var/run/docker.sock      \
  rancher/agent:v0.8.2                              \
  http://[RANCHER_SERVER]/v1/scripts/EFA4EAD......

top and remove the container instance (but not the image)
ker stop $(docker ps -a -q)
ker rm $(docker ps -a -q)
```

we looked at running production deployments safely with zero downtime. We also looked at using DNS to support multi-region o
multi-cloud deployments with automated fail-over. Lastly, we looked at using global scheduling in Rancher in conjunction with Am
caling arrays to build services which are elastic to incoming load. This by no means an exhaustive list of considerations for a
ction environment as each production environment is a unique snow-flake. However, any large-scale production deployment need
hese factors into consideration. In subsequent articles we will look at more considerations that are important for running and
aining Dockerized production deployments especially in statefull workloads such as databases. To get started with Rancher, join
and start building your container service, or download the eBook on building a CI/CD pipeline with Rancher and Docker.

*n and Bilal are server and infrastructure engineers, with experience in building large scale distributed services on top of various c
rms. You can read more of their work at techtraits.com, or follow them on twitter @usman_ismail and @mbsheikh respectively.*

Products

Rancher

RancherOS

Docs

Rancher 2.0 Docs

Rancher 1.6 Docs

RancherOS

Learn

Schedule a demo

Rancher Kubernetes Training

About

About Us

Blog

Careers

Support

Contact

Get Support

Slack

Forums

GitHub

Legal

EULA

Terms of Service Agreement

Privacy

Get the latest news

email address

Submit