Service Discovery via Consul with Amazon ECS

This lab is taken from the blog post https://aws.amazon.com/blogs/compute/service-discovery-via-consul-with-amazon-ecs/

With the advent of modern microservices-based architectures, many applications are now deployed as a set of distributed components. In such architecture, there is a need to configure and coordinate the various applications running in multiple Docker containers across multiple EC2 instances. Amazon EC2 Container Service (Amazon ECS) provides a cluster management framework that handles resource management, task management, and container scheduling. However, many applications need an additional component to manage the relationships between distributed components. The concept of service discovery is used to describe components that facilitate the management of these relationships. Consul by HashiCorp can augment the capabilities of Amazon ECS by providing service discovery for an ECS cluster.

Basic service discovery

Service discovery tools manage how processes and services in a cluster can find and talk to one another. They involve creating a directory of services, registering services in that directory, and then being able to look up and connect to services in that directory.

For example, if your frontend web service needs to connect with your backend web service, it could hardcode the backend DNS, or it can use a service discovery tool to find the backend by name and other metadata and then get the endpoint IP address and port number. This allows components to know about other components in the cluster, whether they are listening on TCP or UDP ports, and to be able to look up and connect to that service by a specific name. Because service discovery is the glue between the components, it is important that it be highly available, reliable, and quickly respond to requests.

Amazon ECS allows you to run and maintain a specified number of instances of a task definition simultaneously; this is called a service. You can also run services behind a load balancer so that the service name is fixed, making it an easy service discovery solution. While this may work for many applications, it doesn't work for all, and you still need to build discovery code into your application (such as listing all services, then using the describe-service call to get the specific endpoint). Another option is to use a purpose-built service discovery framework.

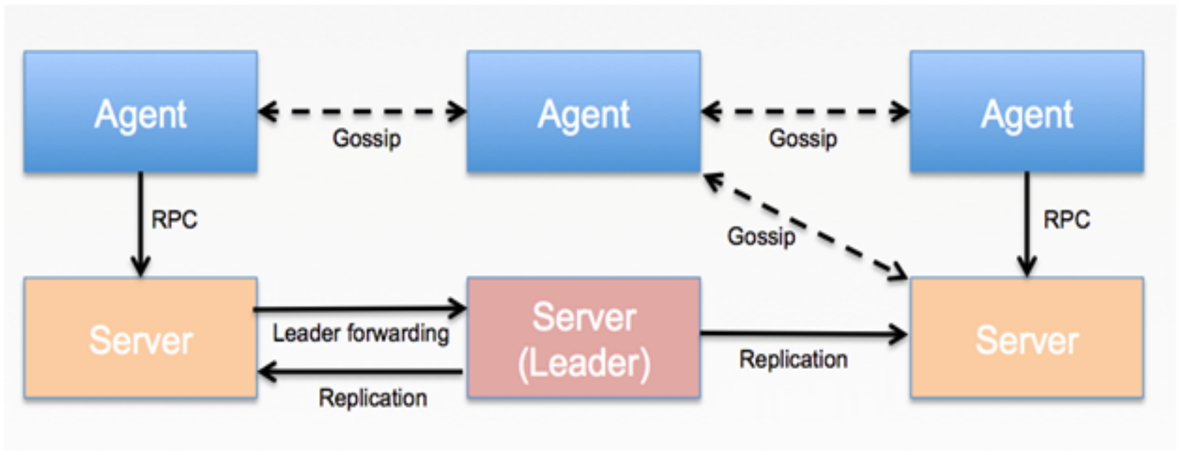
One common tool for service discovery that can overcome some of the challenges outlined above is Consul. It includes health checking as a key component of service discovery and a key/value store that supports application configuration. Consul is a distributed system, allowing it to be highly available, resilient, and performant. One of the benefits is that Consul is easy to integrate with applications via simple HTTP or DNS interfaces for service lookup.

Consul Architecture

Every node that provides services to Consul runs a Consul agent. The agent is responsible for checking the health of the services on the node as well as for the node itself. Checking health is an important part of any service discovery framework, as only healthy services should be discoverable by clients; any unhealthy host is deregistered from the Consul service. Agents communicate with other agents and servers via specific ports and use both TCP and UDP protocols.

The agents talk to one or more Consul servers. The Consul servers store and replicate the data. The servers themselves elect a leader, using the Raft consensus algorithm. While Consul can function with one server, 3 to 5 servers are recommended to avoid failure scenarios leading to data loss. Typically, an application in an AWS region runs a cluster of Consul servers.

Services in your application that need to discover other services can query any of the Consul agents or servers, as agents forward queries to servers. A diagram of the interaction between the Consul agents and servers is shown below.



For more information, see Consul Architecture.

DNS configuration

One of the most powerful features of Consul is that it allows clients to use standard DNS queries to look up services. This can be done with a normal DNS A record lookup or by using DNS SRV record lookups to discover both the IP address and port number of the servers hosting the service. It is useful then that the Docker process is configured in a way that ensures the Consul agent is used as the DNS resolver for all Docker containers running on the same instance.

For example, if the Consul agent is installed on an EC2 instance and is listening on port 53 for DNS requests, you can look up the service named "hello-world" with the following DNS query:

```
$dig @0.0.0.0 –t SRV hello-world.service.consul
```

This returns both the IP address and the port number of the server hosting this service. The answer looks something like the following:

```
;; QUESTION SECTION:
;hello-world.service.consul.   IN      SRV
;; ANSWER SECTION:
hello-world.service.consul.    0  IN    SRV   1 1 80
i-28cdc8ce.node.eu1.consul.
;; ADDITIONAL SECTION:
i-28cdc8ce.node.eu1.consul. 0  IN      A       10.0.1.93
```

Consul and Amazon ECS

Each node that offers a service needs to launch a Consul agent. In the context of Amazon ECS, you need to launch a Consul agent for each ECS instance in the ECS cluster. You can easily do this by containerizing the Consul agent software and launching it via an EC2 UserData script that is invoked when the ECS instance is launched. The agent needs to communicate with a Consul server that stores the service directory. It is possible to launch the Consul server and agent in the same process for testing.

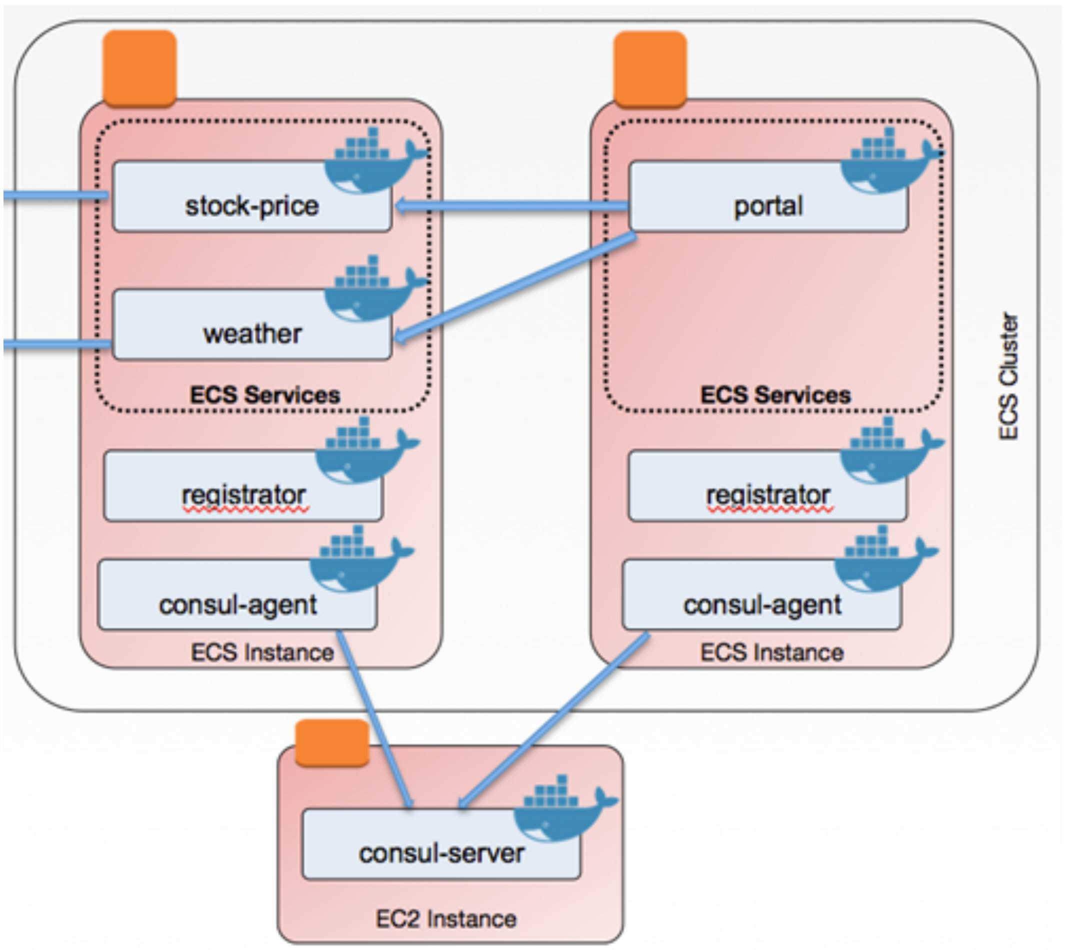
Example application

The best way to explain how Consul and Amazon ECS can work together is through an example application with three components or microservices:

- A "stock-price" service that returns the current stock price and company name for a given stock symbol. The service is implemented as an HTTP service and provides a JSON document in the HTTP response.

- A “weather” service that returns the current temperature of a given city name. The service is implemented as an HTTP service and provides a JSON document in the HTTP response.
- A “portal” service that provides the end-user web page containing the other two services and sends a DNS query that looks for the SRV record type, to get both the service DNS A record name and port number.

A diagram of this architecture is shown below.



In this example architecture, there are two ECS instances running with the Consul agent and an EC2 instance running the Consul server where the Consul service data is stored. This setup is designed for a development & test environment, as there is only one Consul server. In a production setup, 3 to 5 servers are recommended to avoid failure scenarios leading to data loss.

The Amazon ECS cluster is set up to contain the ECS instances that run the microservices. The “stock-price”, “weather”, and “portal” ECS services are deployed to instances running in this ECS cluster. Each ECS instance runs two Docker containers on instance startup, including the Consul agent providing service discovery capabilities via HTTP and DNS to all Docker containers running on the instance. The Consul agent communicates with the Consul server to get the latest state of the cluster.

The other container is a Registrator agent that automatically register/deregisters services for ECS tasks or services based on published ports and metadata from the container environment variables defined in the ECS task definition. It can register services without any user-defined metadata but allows the user to override or customize the Consul service definitions. For more information about Registrator, see the GitHub project.

Run the example application

To follow this example application, the following requirements should be in place in your AWS account:

- A VPC, with DNS support enabled and at least one public subnet
- An IAM user, with permissions to launch EC2 instances and create IAM policies/roles
- An EC2 key pair, for which the user has access to the private key file (*.pem)

You should also have a Docker Hub account and a repository (e.g., "my_docker_hub_repo").

Note: Be sure to input custom values throughout the document as necessary (for example, replace "my_docker_hub_repo" with the name of your own Docker Hub repository).

Create the Amazon ECS cluster

1. Navigate to the ECS console and choose **Create cluster**. Give the cluster a unique name and choose **Create**.

Create ConsulServer and ECS instances

You need to create various AWS resources to get the example application running. To make this easy, there is a CloudFormation script that does the following:

- Creates the IAM role for the ECS and ConsulServer instances.
- Creates the necessary security groups to allow communication between the ECS nodes and the Consul server and allows SSH from a defined IP CIDR range, as well as opening ports 80 and 443 on the Internet.
- Launches the ConsulServer EC2 instance, installs Git and Docker, and launches a Docker container running the Consul server software.
- Launches an Auto Scaling group for the ECS instance cluster with the ecs-optimised AMI.
- Launches the Consul agent as a Docker container and connects to the ConsulServer instance.
- Launches the Consul Registrator agent to register ECS tasks and services launched on the instance automatically with the Consul service discovery directory. The Docker daemon is configured to use the Consul agent as well as the Amazon DNS server for DNS queries.

2. Open the CloudFormation console and launch a new CloudFormation stack from the template provided. You need to enter parameters, including an existing EC2 key pair name, VPC ID, subnet ID, Availability Zone, and so on.

Please note that the input parameter AmazonDnsIp needs to be the DNS server running on a reserved IP address at the base of the VPC network range "plus two". For more information about the Amazon DNS server in a VPC, see the Amazon DNS Server section in the DHCP Options topic.

Build the Docker images

3. Log in via SSH to the ConsulServer public DNS name. This will the value of the output parameter "ConsulServer" from the CloudFormation script launched in the previous step.

4. Download the source code for the three microservices.

You should see three directories that contain the information needed to build three Docker containers.

5. Log in to Docker Hub:

```
$ sudo docker login
```

6. Build the Docker containers in each of the subdirectories, replacing *my_docker_hub_repo* with your repository name:

```
$ cd weather
$ sudo docker build -t my_docker_hub_repo/weather .
$ cd ../stock-price $ sudo docker build -t my_docker_hub_repo/stock-price
.
$ cd ../portal $ sudo docker build -t my_docker_hub_repo/portal .
```

7. Push the images to your Docker Hub repository, replacing *my_docker_hub_repo* with your repository name:

```
$ sudo docker push my_docker_hub_repo/weather
$ sudo docker push my_docker_hub_repo/stock-price
$ sudo docker push my_docker_hub_repo/portal
```

While the Docker images are building and being pushed to Docker Hub, you can view the Dockerfiles in each directory to see what is happening. Here are a few things to note:

- Each container is deployed as a Ruby Sinatra service.
- The weather container application is defined in the ruby file weather.rb. It takes a name of a city via an HTTP GET request, makes an HTTP API call to the open weather map service to get the temperature for the given city, and returns this as a JSON object.
- The stock-price container application is defined in the ruby file stocks.rb. It takes a stock symbol via an HTTP GET request, makes an HTTP API call to the Yahoo Finance service to get the company name and stock price for the given symbol, and returns this as a JSON object.
- The portal application is defined in the ruby file portal.rb. It contains a web page defined in the file public/index.html that has two sections showing a table of stock prices and a table of city temperatures. Stock symbols are entered into the form and the portal application looks up the stock-price service using the Consul service discovery framework via a DNS query for a SRV record. The Consul agent returns the IP address and port number, then the portal application calls the service to get the company name and stock price and displays the result in the table. A similar process occurs for the weather service, whereby the user enters the name of the city and the portal application looks up the IP address and port of the weather service via DNS and make a call to get the temperature of the city.

A code snippet of the Ruby function used in the portal component to do the service discovery lookup is shown below. The function takes a service name, sends a DNS query (looking for an SRV record with that service name), and returns both the IP address and port number of the host running that service.

```
def lookup_service(service_name)
resolver = Resolv::DNS.open
record =
resolver.getresource(service_name,Resolv::DNS::Resource::IN::SRV)
return resolver.getaddress(record.target), record.port
end
```

All services are deployed as ECS services and are automatically registered with the Consul server via the Registrator agent running on each of the ECS instances.

Create the task definitions

Now you need to create the ECS task definitions to be able to launch the previously built containers in your ECS cluster. Open the ECS console to the Task Definition menu.

8. Create the stock price ECS task definition. You can use the following template, replacing *my_docker_hub_repo* with your repository name.

```
{
        "family": "stock-price",
        "containerDefinitions": [
                 {
                         "environment": [
                        {
                         "name": "SERVICE_4567_NAME",
                         "value": "stock-price"
                        },
                        {
                         "name": "SERVICE_4567_CHECK_HTTP",
                         "value": "/health"
                          },
                          {
                          "name": "SERVICE_4567_CHECK_INTERVAL",
                          "value": "10s"
                          },
                          {
                          "name": "SERVICE_TAGS",
                          "value": "http"
                          }
                          ],
                          "name": "stock-price",
                          "image": "my_docker_hub_repo/stock-price",
                          "cpu": 100,
                          "memory": 200,
                          "portMappings": [
                          {
                             "containerPort": 4567
                          }
                          ],
                          "essential": true
                  }
                ]
}
```

It adds metadata that the Registrator agent uses to customise the service definitions with Consul via the environment variables in the ECS task definition, including:

- Setting the name of the service in the Consul server to “stock-price”
- Adding a health check which calls the URL “/health” every 10 seconds
- Adding a service tag of “http”

Notice that you are not defining the host port mapping. Docker automatically assigns a port on the host so the port number is discovered via the Consul service discovery. This allows you to run multiple tasks or services of the same type on a single ECS instance.

9. Create the weather ECS task definition. You can use the following template, replacing *my_docker_hub_repo* with your repository name.

```
{     "family": "weather",     "containerDefinitions": [         {
"environment": [                 {                         "name":
"SERVICE_4567_NAME",                     "value": "weather"
},              {                       "name":
"SERVICE_4567_CHECK_HTTP",                      "value": "/health"
},              {                       "name":
"SERVICE_4567_CHECK_INTERVAL",                      "value": "10s"
},              {                       "name": "SERVICE_TAGS",
"value": "http"               }            ],           "name":
"weather",              "image": "my_docker_hub_repo/weather",
"cpu": 100,              "memory": 200,              "portMappings": [
{                     "containerPort": 4567                 }
],             "essential": true          }      ] }
```

It adds metadata that the Registrator agent uses to customise the service definitions with Consul via the environment variables in the ECS task definition, including:

- Setting the name of the service in the Consul server to "weather"
- Adding a health check which calls the URL "/health" every 10 seconds
- Adding a service tag of "http"

10. Create the portal ECS task definition. You can use the following template, replacing *my_docker_hub_repo* with your repository name.

```
{     "family": "portal",     "containerDefinitions": [         {
"name": "portal",             "image": "my_docker_hub_repo/portal",
"cpu": 100,             "memory": 200,             "portMappings": [
{                      "containerPort": 4567,
"hostPort": 80                  }              ],              "essential":
true          }     ] }
```

Create the ECS services

11. On the **Services** tab in the ECS console, choose **Create**. Choose the task definition created in step 7, name the service, and set the number of tasks to 1. Select **Create service**.

12. Repeat step 11 for the tasks created in steps 8 and 9 to launch the stock-price and portal services.

13. The services will start running shortly. You can press the refresh icon on your service's **Tasks** tab. After the status of the portal service says "Running", choose the task and expand the portal container. The portal container instance IP is a hyperlink under the **Network bindings** section **External link** field. Open the URL of the portal service.

14. You can now enter stock symbols into the text box and the portal does a lookup of the stock-price service and gets the latest stock price and company name. You can also enter a name of a city in the Weather section of the page and it does a lookup of the weather service to get the latest temperature in Celsius.

Conclusion

If you have followed all of these steps, you should now have three containers running in your ECS cluster. One container hosts the weather service that returns the temperature for a city. Another hosts the stock-price service that returns the stock price and company name of a given stock symbol. The last container hosts the portal application that provide a web page for end-users to look up the weather and stock prices via the Consul service discovery agents deployed on the ECS instance.

View the source code of the containers to replicate this setup and build your own microservices architecture with the service discovery feature.