Automated Safeguards for ECS (PHI Containers)

The ClearDATA PHI Container Cluster is designed to provide a secure, compliant microservice platform using of ClearDATA's Dynamic PHI and Amazon's EC2 Container Service (ECS). The platform extends ECS in order to provide a compliant-by-default system with a smooth developer experience. The key extensions include:

- Responsive service discovery
- Integrated request routing
- ClearDATA hardened instances
- Encrypted overlay network
- Encrypted instance volumes
- Coverage under ClearDATA's BAA available

Technical overview

This section provides an overview of the platform, in particular as it applies to running web services.

Underlying infrastructure

The cluster consists of EC2 instances deployed by an Auto Scaling Group. Auto Scaling ensures that the minimum number of instances is healthy at all times, and automatically replaces any instances that go offline for any reason.

Each EC2 instance is automatically configured on its first boot to run the software required by ClearDATA. That process takes a few minutes. At the end of that process, the instance registers itself with ECS, and is ready to run containerized applications.

ECS Services and Tasks

An ECS *service* is a long-running piece of software like a web service API. The service configuration specifies a number of copies to run. ECS tries to ensure that the correct number of copies are always available. If one exits unexpectedly, a new copy will automatically replace it.

An ECS *task* runs a piece of software once, and does no subsequent monitoring. If it exits, the task is complete. Tasks are more appropriate for periodic, batch, or event processing systems.

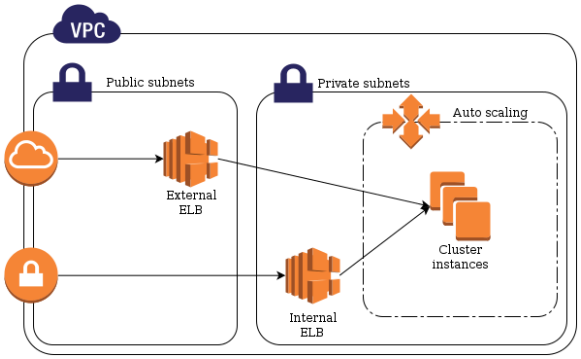
Note

One special use-case to be aware of: when ECS runs a service, it does this by running a number of ECS tasks. For more info on services and tasks see: Services and Running Tasks.

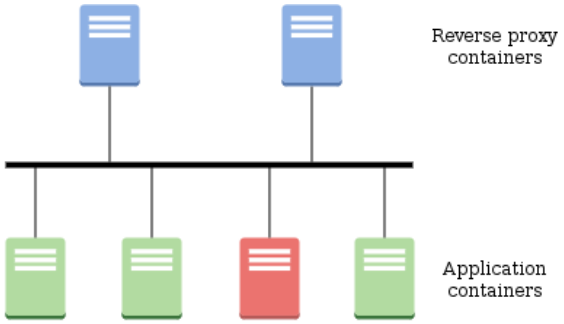
Request routing

To provide a containerized web service, HTTP requests must eventually be delivered to the application that can handle them.

When running a web service, HTTP requests needs to eventually be delivered to the application container. The first step is be delivered to a cluster instance:



Incoming requests will be routed to the instances by one or more Elastic Load Balancers over HTTPS. Each instance has an automatically configured reverse proxy which routes the requests to one of the containers for the requested service:



When a request arrives at an EC2 instance, the reverse proxy container on that instance is responsible for routing it to the appropriate kind of application container. Any EC2 instance can handle requests for any application container. If the ECS service launches multiple copies of an application, the reverse proxy container will load balance among them.

For example, the above diagram shows three copies of the green application running on the cluster. Both reverse proxy containers can handle requests for the green application. Requests will be load balanced among the three green containers.

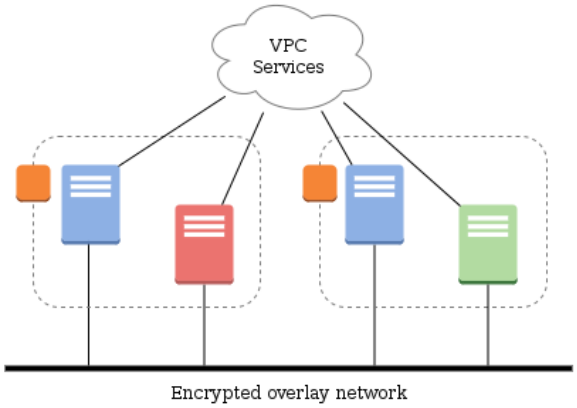
Overlay network

In a service-oriented architecture, it's desirable to make services that can easily be accessed by other services. This permits creating simpler, isolated services which can be combined to provide reliable and scalable applications.

To help enable applications that process PHI, the PHI Container Cluster connects each container to a private overlay network. This allows containers to directly communicate with one another, regardless of which cluster instances they are on.

This overlay network is built to ensure that all communications are always encrypted in transit. Normally, if an application processes PHI, it must ensure that the protected data is always encrypted when being transmitted in an AWS VPC. By building in an encrypted overlay network, the cluster can provide that encryption transparently.

In an application container, the overlay network looks like a secondary network interface:



The overlay network is assigned 10.32.0.0/12 only. The default gateway remains configured in the usual way. This ensures that it does not affect ordinary internet, VPC, or VPN traffic.

> This means that, by default, containers will be unable to communicate with any infrastructure outside of the cluster that is numbered out of 10.32.0.0/12. This includes anything in an Amazon VPC, as well as infrastructure reached over a VPN, Direct Connect, VPC peering, etc. If you need to reach infrastructure in 10.32.0.0/12 the overlay subnet must be changed. Please work with ClearDATA Support to request this.

Responsive service discovery

When a container starts on a cluster, software running on the EC2 instance automatically registers that new container and provides information on it to the other EC2 instances in the cluster. That process is local to each instance, and proceeds as follows:

1. The ECS agent starts the new container.
2. The service discovery registration agent receives a notification from Docker with information on the new container.
3. Information is registered in the service discovery system.
4. If required, the reverse proxies are configured with the relevant information about the new container.

For details on how to use the service discovery, expose services using these tools, and to control what is registered, see Service discovery.

Customizations

Many of the parameters and behaviors here can be customized as required. Some possible examples include:

- Some clusters might not handle requests from the internet. Such a cluster has no need for an External ELB.
- Different auto scaling behavior may be desirable for different clusters. The three instance minimum size cannot be avoided, but the maximum number and scaling behavior can be customized.
- The overlay network used to enable encrypted container-to-container communication can be extended to additional instances. This can be used to expose a database on an EC2 instance to containers over the encrypted network connection.

Please contact your account team or ClearDATA Support for help in making such customizations.

Platform Recommendations

The ClearDATA PHI Container Cluster uses a variety of AWS products to provide dynamic, scalable, and reliable platform services that are HIPAA compliant. This means that some application architectures will work better than others. Some guidelines for an optimal experience are below.

Stateless container applications

Containers should never store important state information inside their local filesystem. There are a number of reasons for this:

1. If the container is recreated, any changes to its filesystem are lost. Relying on container storage for important data will lead to data loss.
2. Requests may be routed to any healthy container providing a given service. Imagine that a client makes a request, and the container records some state data on its local filesystem. Subsequent requests from that client may be routed to other containers, which will not have the state data.
3. If an application has no local state, it can be easily scaled up by deploying more copies. Those new copies will be ready to serve requests without syncing any data. Similarly, scaling down is simple, as there's no local data to be saved.

External storage

In order to avoid storing local state, applications that you intend to run on a PHI Container Cluster should store all of their data in an external storage system. S3, RDS, and DynamoDB are often good choices.

> ⚠ When storing PHI, only use BAA covered storage systems.

In standard ClearDATA deployments, a PHI Container Cluster can access AWS services so no special handling is required. A containerized application can use the standard SDKs and APIs to access AWS storage systems like the above.

Of course that requires AWS credentials to be configured. See Per-container IAM roles for information on our recommended method.

> ⛔ Since IAM users have static keys, ClearDATA *strongly* discourages their use in this setting.

Observe container resource usage

By default, ECS automatically attempts to schedule containers on instances by comparing the task definition's requirements to the resources available on each instance. This include the memory and CPU values from the task definition. Optimal use of a PHI Container Cluster requires reasonable estimates of a container's memory and CPU usage.

If a container exceeds its memory hard limit, it is killed. Ideally, a container's memory limit should be high enough to provide memory for normal operation, while still being as low as possible. Practically, it is best to start higher, and lower the limit as more memory usage data helps with your estimation.

CPU usage is advisory - a task will not be killed if it consumes more CPU than allocated in the task definition. However, ECS uses that value to determine if a task will fit on a container instance. Ideally, it'd be best to set this to the average usage of the container, but this is difficult to determine. Practically, it is best to start low and increase as more CPU usage data helps with your estimation.

Resources

There are many resources on container-centric application development. Some resource worth looking at include:

- Amazon EC2 Container Service Documentation
- Docker Documentation
- The Twelve-Factor App