Automating PeopleSoft Environments in the AWS Cloud

April 2019

Notices

Contents

Abstract

This whitepaper describes an approach for automating the creation of Oracle PeopleSoft (PeopleSoft) environments in the Amazon Web Services (AWS) Cloud. This approach uses currently available AWS products and services whenever possible, and uses AWS recommended tools to migrate client data and construct the environment-creating automations. The approach has a material impact on the flexibility, consistency, stability, scalability, security, and cost of PeopleSoft environments in the AWS Cloud.

Automations Approach

Numerous enterprises have successfully migrated their Oracle PeopleSoft (PeopleSoft) applications to the Amazon Web Services (AWS) Cloud. The majority of these migrations have been *lift-and-shift* projects, which are projects in which a static architecture, similar to the enterprise’s previous on-premises-based deployment, is recreated in the AWS Cloud. Such enterprises have effectively used the AWS infrastructure as a service (IaaS) offering to outsource their data centers to the AWS Cloud.

This approach fails to fully benefit from the on-demand capabilities of AWS services. The suggested approach outlined in this paper uses currently available AWS products and services, with AWS-supported automation tools, to build fully automated PeopleSoft environment-creating solutions (the Automations). After the enterprise’s data has been migrated to the AWS Cloud, new PeopleSoft environments can be created in about 40 minutes. This includes restoring the database, building out and configuring the web, app, and batch servers and the accompanying file systems, and properly integrating the components all within a virtual private cloud in AWS.

The Automations have a material impact on numerous operating attributes of PeopleSoft environments in the AWS Cloud.

Benefits of Automating PeopleSoft Environments in the AWS Cloud

Automating PeopleSoft environments in the AWS Cloud has several very important benefits, including the following.

1. **Flexibility** – Because the environments are created in about 40 minutes, non-production environments can be destroyed each night and rebuilt each morning. New environments can be created for projects quickly and then terminated when the project is completed. This flexibility means that enterprises can be constantly right-sizing their solutions and can quickly react to changing resource needs.
2. **Consistency** – Environment creation is automated, which makes the environments extremely consistent throughout the application stack. Each environment is built from the same set of build scripts. This leads to highly predictable operations and results.
3. **Change Control** – Because the environments use infrastructure as code, all changes to configuration or installation are tracked through source control. This makes investigation into why an environment has a certain configuration much easier to track.
4. **Stability** – The environments are very stable because the environments have been thoroughly tested and are created in exactly the same way each time.
5. **Scalable** – Using delivered scaling services, such as AWS Auto Scaling and Amazon ECS, environments can be quickly scaled up or down, on demand. This allows the environments to be very reactive to changing resource utilization requirements. Environments can be scaled up on demand to meet the resource needs of high traffic events, such as student registration or period-end reporting. The environments can then be scaled down to normal resource levels as needed. When you adopt this approach, your environments do not need to be over provisioned to meet peak usage and peak performance. Instead, they can be more elastic to respond to real-time demands.
6. **Security** – AWS assets are available to root administrators only upon creation, which means access is limited entirely to the security policy that is implemented during the build out. By extension, this means that security and access are driven entirely by the security policies reflected in the environment-building automations. Reviewing and revising an implemented security policy becomes much easier with such centralized control.

7. **Economics** – Because AWS charges hourly for their services, and because environments can be launched and scaled so quickly, clients can regularly right-size their environments for ongoing or one-time needs. This also enables clients to establish policies for terminating non-production environments on nights and weekends. This flexibility saves money by reducing the AWS services consumed to only what is required at any given time.

These Automations have a material, positive impact on an enterprise’s ability to successfully and economically operate PeopleSoft environments in the AWS Cloud.

Amazon Web Services Functionality in Use

This automated solution uses the AWS services described in the following sections.[1]

Network

Amazon Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost-effective method to route end users to internet applications. DNS translates human readable names (such as www.example.com) into the numeric IP addresses (such as 192.0.2.1) that computers use to connect to each other.

Amazon VPC

Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways. You can use both IPv4 and IPv6 addresses in your Amazon VPC for secure and easy access to resources and applications. You can easily customize the network configuration for your Amazon VPC.

For example, you can create a public-facing subnet for your web servers that has access to the internet, and place your backend systems, such as databases or application servers, in a private-facing subnet with no internet access. You can leverage multiple layers of security (including security groups and network access control lists) to help control access to Amazon EC2 instances in each subnet. Additionally, you can create a hardware virtual private network (VPN) connection between your corporate data center and your Amazon VPC, and leverage the AWS Cloud as an extension of your corporate data center.

Compute

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.

Amazon ECR

Amazon Elastic Container Registry (Amazon ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images. Amazon ECR is integrated with Amazon Elastic Container Service (Amazon ECS), which simplifies the development to production workflow.

Amazon ECS

Amazon EC2 Container Service (Amazon ECS) is a highly scalable, high-performance container management service that supports Docker containers. It enables you to easily run applications on a managed cluster of Amazon EC2 instances. Amazon ECS eliminates the need for you to install, operate, and scale your own cluster management infrastructure. With simple API calls, you can launch and stop Docker-enabled applications, query the complete state of your cluster, and access many familiar features such as security groups, Elastic Load Balancing, Amazon Elastic Block Store (Amazon EBS) volumes, and AWS Identity and Access Management (IAM) roles. You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs and availability requirements. You can also integrate your own scheduler or third-party schedulers to meet specific business or application requirements.

ELB

Elastic Load Balancing (ELB) automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic.

Storage

Amazon EBS

Amazon Elastic Block Store (Amazon EBS) provides persistent block storage volumes for use with Amazon EC2 instances in the AWS Cloud. Each Amazon EBS volume is automatically replicated within its Availability Zone to protect you from component failure, offering high availability and durability. Amazon EBS volumes offer the consistent and low-latency performance needed to run your workloads. With Amazon EBS, you can scale your usage up or down within minutes—all while paying a low price for only what you provision.

Amazon EFS

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with Amazon EC2 instances in the AWS Cloud. Amazon EFS is easy to use and offers a simple interface that allows you to create and configure file systems quickly and easily. With Amazon EFS, storage capacity is elastic, growing and shrinking automatically as you add and remove files, so your applications have the amount of storage they need, when they need it.

Amazon S3

Amazon Simple Storage Service (Amazon S3) is object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web. It is designed to deliver 99.999999999% durability, and scales past trillions of objects worldwide.

Database

Amazon RDS

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, which frees you up to focus on your applications and business. Amazon RDS provides you six familiar database engines to choose from, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server.

Automation Tools in Action

Docker

This automation process relies heavily on the Docker[2] open source tool. Docker allows components of the PeopleSoft applications to be containerized. This means they are run in isolation from the host operating system and other containers that run on the host machine. It also means that because the application is containerized, deployment and orchestration automation can be developed in a standard way. The Automations leverage existing infrastructure automations from AWS and use Amazon ECS instead of building custom automations to provision and scale applications and environments.

In some ways, Docker is similar to virtual machine software. Like virtual machine software, Docker automates the creation of a well-defined and consistently deployed workspace. Unlike virtual machine software, Docker accomplishes this by sharing the host operating system kernel and using kernel functionality to create an isolated workspace.

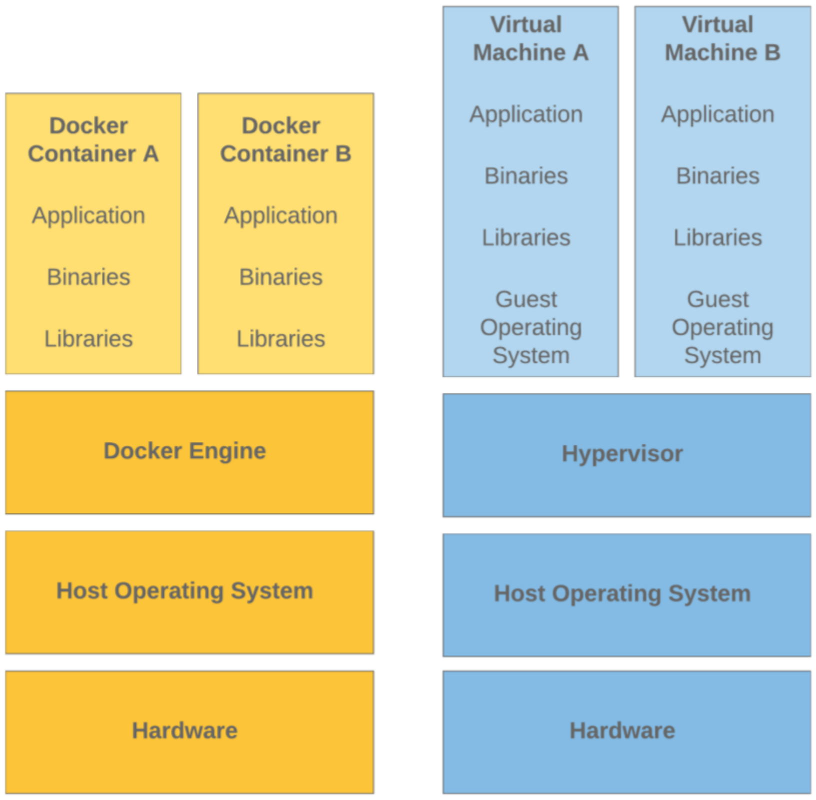
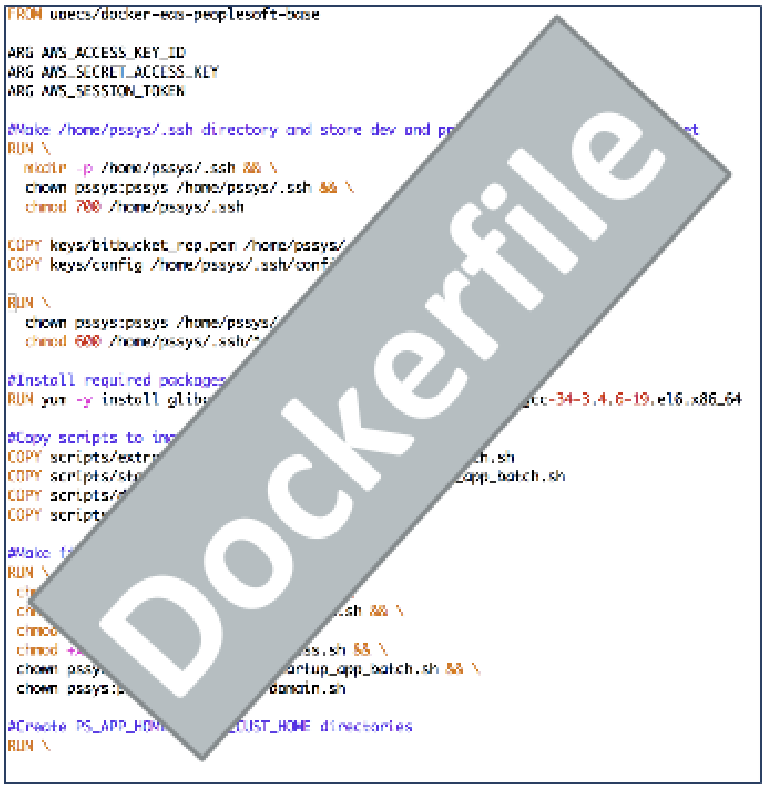


Figure 1 – Docker container architecture vs. virtual machine architecture

Docker uses the kernel in the host machine to create an isolated workspace, known as a *container*, where an application and its dependencies can run consistently and reliably.

The process to create a container begins with a *Dockerfile*, which is a script written in the Go programming language, composed of various commands and arguments listed successively, which perform actions on an existing, base Docker image to create a new, more specialized Docker image.



```
FROM uoecs/docker-aws-peoplesoft-base

ARG AWS_ACCESS_KEY_ID
ARG AWS_SECRET_ACCESS_KEY
ARG AWS_SESSION_TOKEN

#Make /home/pssys/.ssh directory and store dev and pr[illegible]et
RUN \
  mkdir -p /home/pssys/.ssh && \
  chown pssys:pssys /home/pssys/.ssh && \
  chmod 700 /home/pssys/.ssh

COPY keys/bitbucket_rep.pem /home/pssys/[illegible]
COPY keys/config /home/pssys/.ssh/conf[illegible]

RUN \
  chown pssys:pssys /home/pssys/[illegible]
  chmod 600 /home/pssys/.ssh/[illegible]

#Install required packages[illegible]
RUN yum -y install glib[illegible]gcc-34-3.4.6-19.el6.x86_64

#Copy scripts to im[illegible]
COPY scripts/extr[illegible]ch.sh
COPY scripts/st[illegible]_app_batch.sh
COPY scripts/[illegible]
COPY script[illegible]

#Make f[illegible]
RUN \
 c[illegible]
 c[illegible]sh && \
 chmod[illegible]
 chmod +[illegible]s.sh && \
 chown pssy[illegible]artup_app_batch.sh && \
 chown pssys:[illegible]domain.sh

#Create PS_APP_HOM[illegible]CUST_HOME directories
RUN \
```

Figure 2 – Example of a Dockerfile

The *Docker Engine* uses the Docker `build` command to convert the commands and arguments in a Dockerfile into an image. The Docker Engine comprises a command line interface (CLI), an API, and a daemon. The daemon is the long running software that manages Docker images and containers.

An *image* is a binary file created by the Docker Engine using the `build` command and the instructions in the Dockerfile.[3] The image is based on the Open Container Initiative (OCI) standard format. The image file contains an ordered collection of root filesystem changes (the filesystem changeset) and the corresponding execution parameters (described in JSON format) for use within a container runtime (the container component of the Docker Engine).[4] Each pair of changesets and parameters creates a layer within the image. Each layer is a read-only file generated by running a command from a Dockerfile.[5]

A container is an isolated workspace that runs an application, including its dependencies. The container is created by the Docker Engine using the `run` command on an image. The `run` command creates the container, layer-by-layer, based on the image. Each image layer is read-only and, when complete, forms the container root filesystem. When all of the image layers are assembled, a final read/write layer is added to record the changes made in the container during a session. A container can also be viewed as a running or instantiated image, with a final read/write layer added.

AWS CloudFormation

AWS CloudFormation is an AWS service through which users can create a template that describes all the AWS infrastructure resources needed to create user-defined services. AWS CloudFormation takes care of provisioning and configuring all the specified infrastructure components and services. The AWS CloudFormation template describes exactly what resources are provisioned and their settings. Templates are text files, so users can see the differences in template versions and track changes in their infrastructure solution. [6]

Amazon ECS

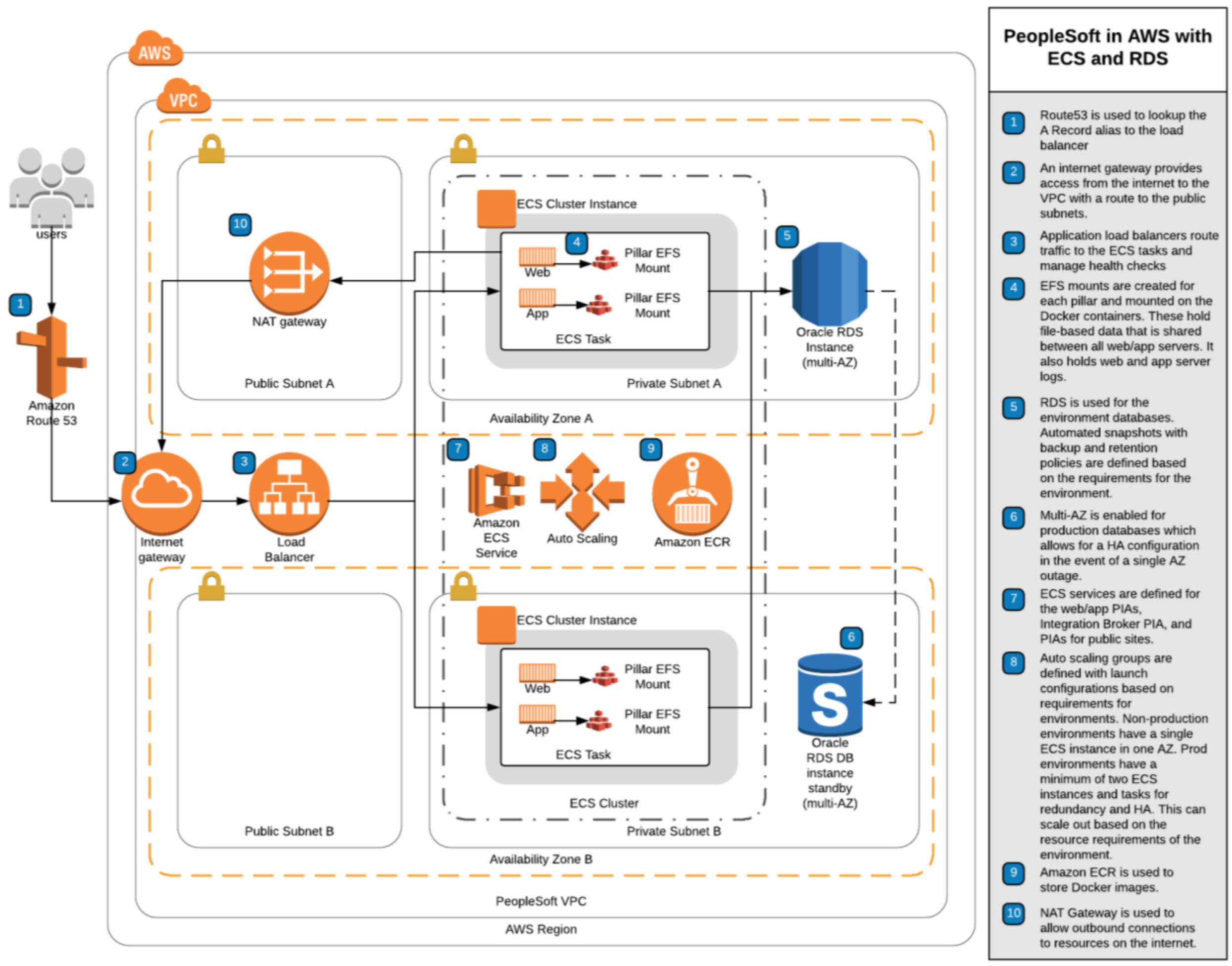
Amazon Elastic Container Service (Amazon ECS) is a scalable, container management solution that supports Docker containers. Amazon ECS schedules, runs, and scales containerized applications on AWS. Amazon ECS can launch and stop Docker-enabled applications, query the complete state of such applications, and access many AWS features, such as IAM roles, security groups, load balancers, and AWS CloudFormation templates.[7]

Amazon RDS

Amazon Relational Database Service (Amazon RDS) is a web service that simplifies the setup, operation, and scaling of a relational database in the AWS Cloud. Amazon RDS provides cost-efficient, resizable capacity for an industry-standard relational database, and manages common database administration tasks.[8]

Environment Architecture

The application architecture of a PeopleSoft environment in the AWS Cloud was designed to be scalable for production environments and flexible for smaller workloads of non-production environments. Using AWS CloudFormation and Amazon ECS ensures that each environment is deployed and configured in a consistent manner. Below is a high-level architecture of all of the components that are deployed as part of the AWS CloudFormation deployments for each environment.



PeopleSoft in AWS with ECS and RDS

1 Route53 is used to lookup the A Record alias to the load balancer

2 An internet gateway provides access from the internet to the VPC with a route to the public subnets.

3 Application load balancers route traffic to the ECS tasks and manage health checks

4 EFS mounts are created for each pillar and mounted on the Docker containers. These hold file-based data that is shared between all web/app servers. It also holds web and app server logs.

5 RDS is used for the environment databases. Automated snapshots with backup and retention policies are defined based on the requirements for the environment.

6 Multi-AZ is enabled for production databases which allows for a HA configuration in the event of a single AZ outage.

7 ECS services are defined for the web/app PIAs, Integration Broker PIA, and PIAs for public sites.

8 Auto scaling groups are defined with launch configurations based on requirements for environments. Non-production environments have a single ECS instance in one AZ. Prod environments have a minimum of two ECS instances and tasks for redundancy and HA. This can scale out based on the resource requirements of the environment.

9 Amazon ECR is used to store Docker images.

10 NAT Gateway is used to allow outbound connections to resources on the internet.

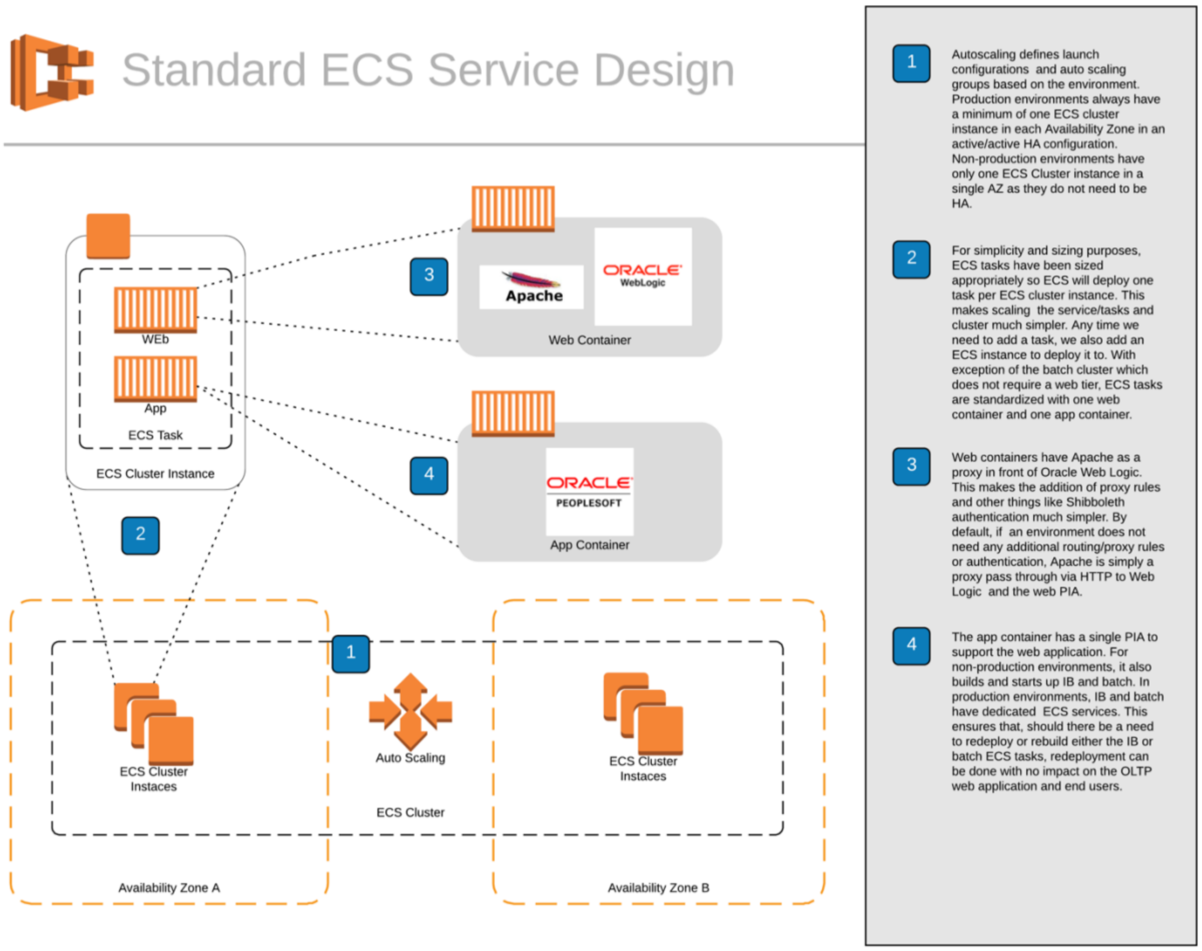
Figure 3 – Architecture for PeopleSoft in AWS with Amazon ECS and Amazon RDS

The AWS CloudFormation templates create the VPC (virtual private cloud), load balancers, Amazon EC2 servers, Amazon ECS clusters, and Amazon RDS instances, that form the environments. The Amazon VPC restricts access to the environments to only customer-approved users.

Automation scripts and AWS CloudFormation are used to generate two types of deployments: one for the production environment and one for the non-production environments. The automation scripts for both the production and non-production environments use all of the same components, however, they are composed and behave differently.

Amazon ECS Service Design

Amazon ECS orchestrates Docker containers and creates tasks, or groups of container services, that are essential elements of the environments. Amazon ECS also monitors the health of these tasks and launches new tasks if an issue with an existing task is detected. The Docker images in Amazon ECR and managed by Amazon ECS contain the instructions necessary to build out the web, application, and batch containers. For example, the Amazon ECS task definition might combine a web and application container into a single task which can be monitored and managed by Amazon ECS.



1 Autoscaling defines launch configurations and auto scaling groups based on the environment. Production environments always have a minimum of one ECS cluster instance in each Availability Zone in an active/active HA configuration. Non-production environments have only one ECS Cluster instance in a single AZ as they do not need to be HA.

2 For simplicity and sizing purposes, ECS tasks have been sized appropriately so ECS will deploy one task per ECS cluster instance. This makes scaling the service/tasks and cluster much simpler. Any time we need to add a task, we also add an ECS instance to deploy it to. With exception of the batch cluster which does not require a web tier, ECS tasks are standardized with one web container and one app container.

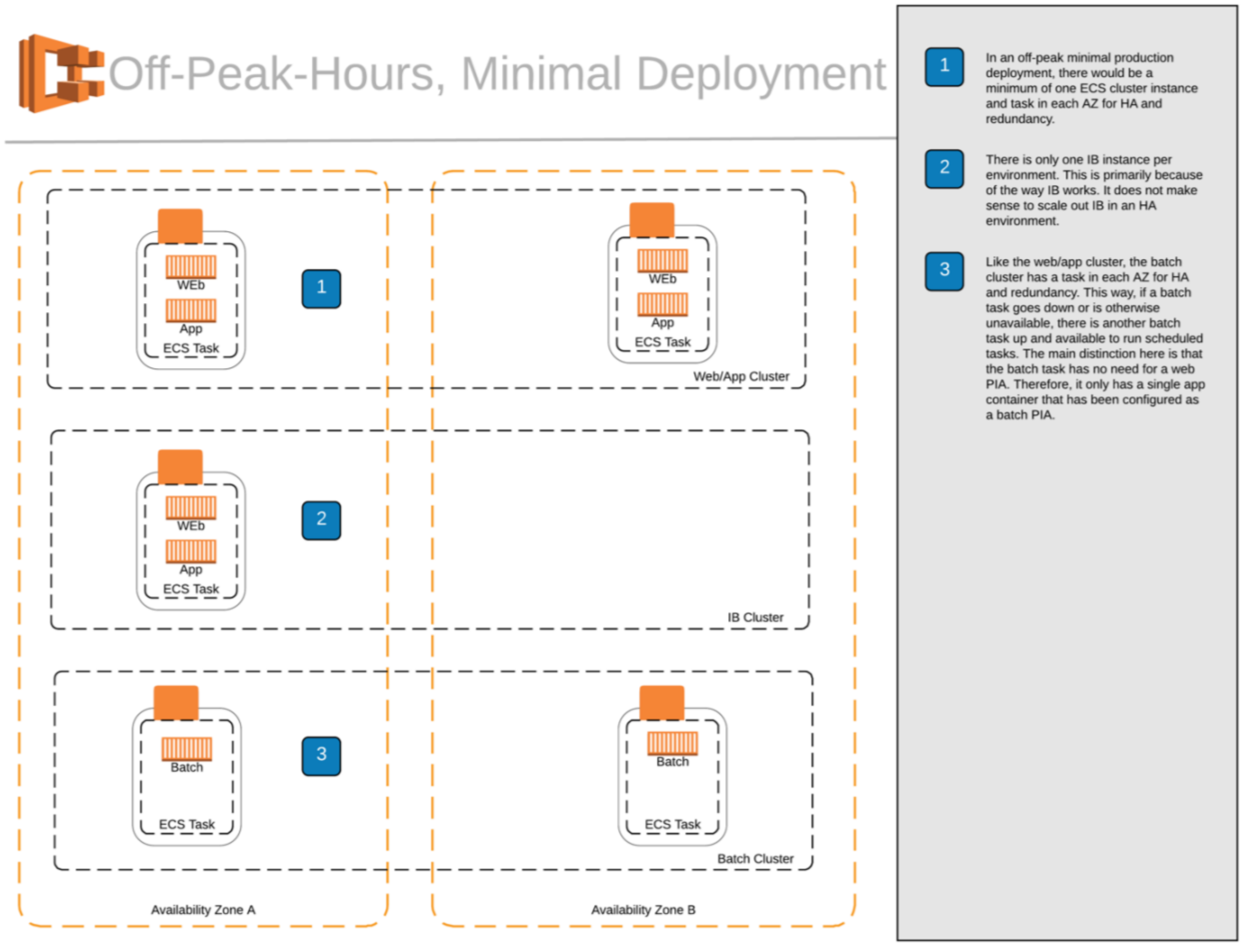
3 Web containers have Apache as a proxy in front of Oracle Web Logic. This makes the addition of proxy rules and other things like Shibboleth authentication much simpler. By default, if an environment does not need any additional routing/proxy rules or authentication, Apache is simply a proxy pass through via HTTP to Web Logic and the web PIA.

4 The app container has a single PIA to support the web application. For non-production environments, it also builds and starts up IB and batch. In production environments, IB and batch have dedicated ECS services. This ensures that, should there be a need to redeploy or rebuild either the IB or batch ECS tasks, redeployment can be done with no impact on the OLTP web application and end users.

Figure 4 – Example of an Amazon ECS service design

Scalable Production Deployment

Production environment deployments are built to allow the OLTP (online transaction processing) application, process scheduler, and integration brokers to be decoupled into separate Amazon ECS services. This allows each component to be scaled independently. The following are some examples of methods you can use to scale your production environments up or down to meet your business needs.



1 In an off-peak minimal production deployment, there would be a minimum of one ECS cluster instance and task in each AZ for HA and redundancy.

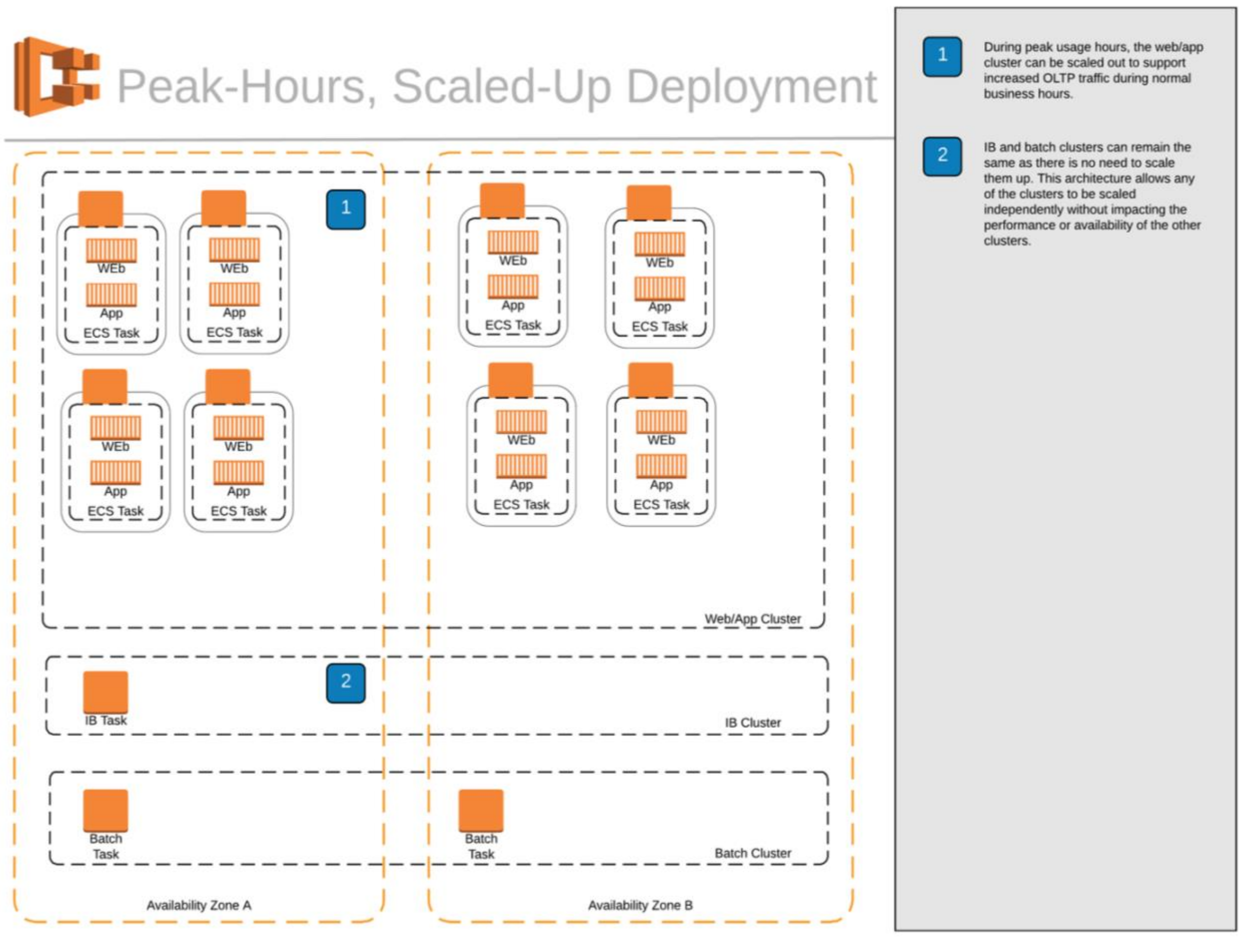
2 There is only one IB instance per environment. This is primarily because of the way IB works. It does not make sense to scale out IB in an HA environment.

3 Like the web/app cluster, the batch cluster has a task in each AZ for HA and redundancy. This way, if a batch task goes down or is otherwise unavailable, there is another batch task up and available to run scheduled tasks. The main distinction here is that the batch task has no need for a web PIA. Therefore, it only has a single app container that has been configured as a batch PIA.

Figure 5 – Example of an off-peak hours, minimal production deployment

In off-peak hours, a production environment can be scaled down to a minimal footprint. This allows the application to continue to be available, but takes advantage of cost savings by only running a minimal set of resources.

In this example, there are three Amazon ECS service clusters defined. This allows each component to be scaled independently. For example, the following diagram shows how this architecture can be scaled up from a minimal footprint to handle regular, business-hour traffic in the web application.



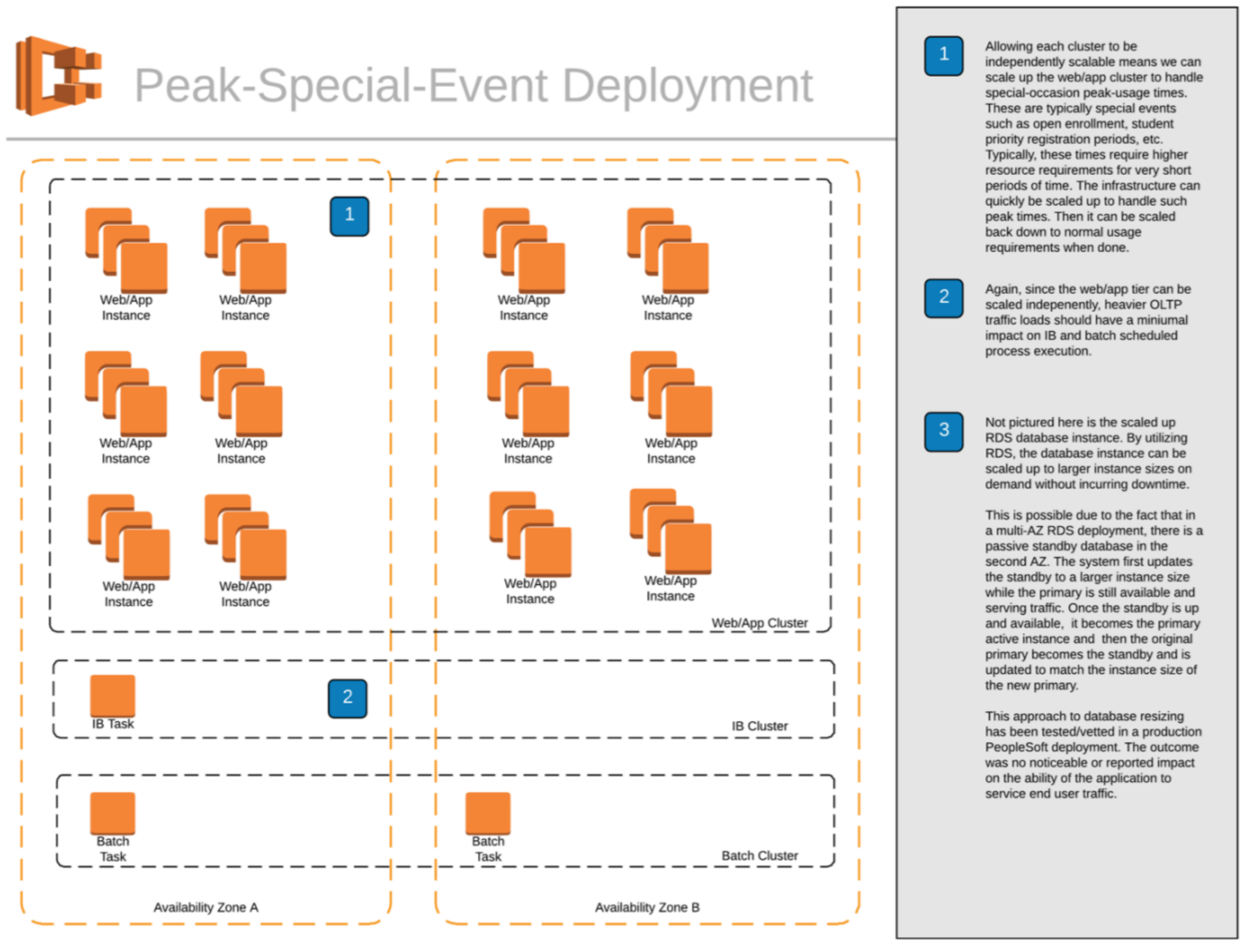
1 During peak usage hours, the web/app cluster can be scaled out to support increased OLTP traffic during normal business hours.

2 IB and batch clusters can remain the same as there is no need to scale them up. This architecture allows any of the clusters to be scaled independently without impacting the performance or availability of the other clusters.

Figure 6 – Example of a peak-hour, scaled-up deployment

In this example, the web/app service cluster has been scaled up to support more end users. It is not necessary to scale up the IB tasks or batch clusters, so they maintain a minimal footprint.

This architecture can be scaled up even further to handle peak usage during special events. Examples for such special events are priority registration times for Campus Solutions or open enrollment for Human Capital Management. This architecture is designed to handle such events by only scaling the components required to handle those special events. The following is an example of a peak, special event deployment footprint.



Allowing each cluster to be independently scalable means we can scale up the web/app cluster to handle special-occasion peak-usage times. These are typically special events such as open enrollment, student priority registration periods, etc. Typically, these times require higher resource requirements for very short periods of time. The infrastructure can quickly be scaled up to handle such peak times. Then it can be scaled back down to normal usage requirements when done.

Again, since the web/app tier can be scaled indepenently, heavier OLTP traffic loads should have a miniumal impact on IB and batch scheduled process execution.

Not pictured here is the scaled up RDS database instance. By utilizing RDS, the database instance can be scaled up to larger instance sizes on demand without incurring downtime.

This is possible due to the fact that in a multi-AZ RDS deployment, there is a passive standby database in the second AZ. The system first updates the standby to a larger instance size while the primary is still available and serving traffic. Once the standby is up and available, it becomes the primary active instance and then the original primary becomes the standby and is updated to match the instance size of the new primary.

This approach to database resizing has been tested/vetted in a production PeopleSoft deployment. The outcome was no noticeable or reported impact on the ability of the application to service end user traffic.

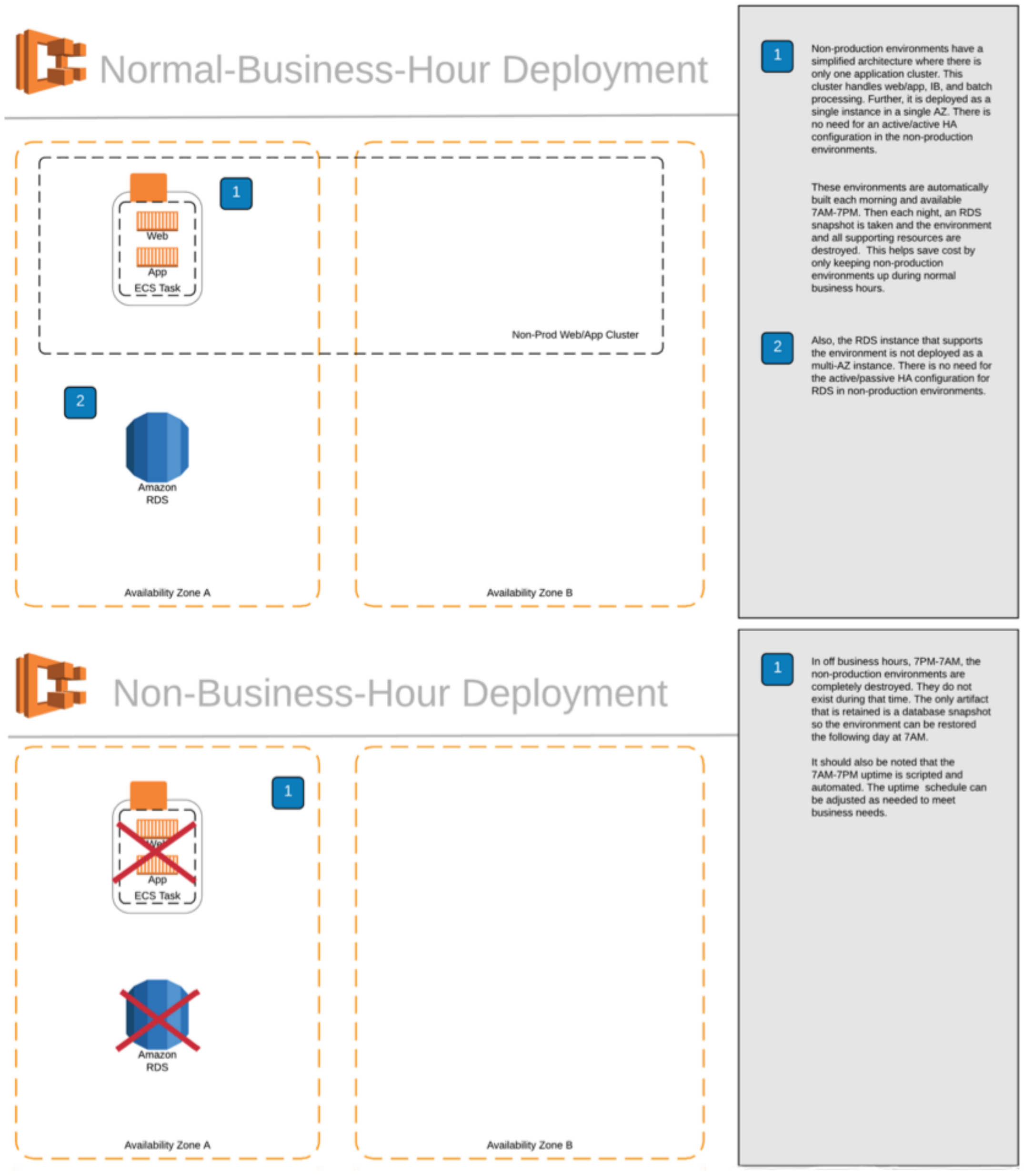
Figure 7 – Example of a peak, special event deployment

Scalable Production Deployment

Non-production environments are designed differently than production environments. They do not separate web/app instances, IB tasks, and integration broker into separate services because each component does not need to be scaled independently. The services also do not span multiple Availability Zones (Multi-AZ) because a high availability configuration is not necessary.

Similar to off-peak hours, peak-hours, and peak, special event times in production environments, non-production environments can be configured to only use the resources required at the time to meet business needs. Non-production environment usage can be configured into normal-business-hour and non-business-hour deployments. For example, you might want all non-production environments to be available from 7:00 AM to 7:00 PM, Monday through Friday. In non-business hours, you might not need to access those resources, so, from 7:00 PM to 7:00 AM, Monday through Friday and on weekends, the environments can be terminated so you are not paying for resources that are not in use.

The following is an example of how this architecture can be leveraged to support normal-business-hour and non-business-hour deployments for non-production environments.



1 Non-production environments have a simplified architecture where there is only one application cluster. This cluster handles web/app, IB, and batch processing. Further, it is deployed as a single instance in a single AZ. There is no need for an active/active HA configuration in the non-production environments.

These environments are automatically built each morning and available 7AM-7PM. Then each night, an RDS snapshot is taken and the environment and all supporting resources are destroyed. This helps save cost by only keeping non-production environments up during normal business hours.

2 Also, the RDS instance that supports the environment is not deployed as a multi-AZ instance. There is no need for the active/passive HA configuration for RDS in non-production environments.

1 In off business hours, 7PM-7AM, the non-production environments are completely destroyed. They do not exist during that time. The only artifact that is retained is a database snapshot so the environment can be restored the following day at 7AM.

It should also be noted that the 7AM-7PM uptime is scripted and automated. The uptime schedule can be adjusted as needed to meet business needs.

Figure 8 – Examples of non-production environment deployments

Development in Automated Environments

To use the Automations to rapidly create production and non-production PeopleSoft environments, you must make changes to the development process for the enhancements. First, make a distinction between enhancements that reside in the database and those that reside outside it.

Enhancements that reside inside the database are captured in the database image when it is created and archived when it is terminated. Those enhancements are created again when you relaunch the non-production database. These enhancements do not cause issues for the Automations.

Enhancements that reside in a file system that is outside the database must be captured and managed separately through processes other than database artifacts. Such enhancements include custom or customized SQLs, customized COBOLs, and custom Java classes.

The Automations rely on two open source solutions—Bitbucket and Git—to store and manage the filesystem enhancements, and to make them available to the Automations. Git is an open-source version control solution. Bitbucket is a Git repository, which is essentially a folder with successively saved or committed versions of a project. Git manages the versions and Bitbucket stores the versions. The Automations pull the relevant customizations into the associated environment implementations through Git commands.

The varying states of the non-production environment’s file-based enhancements are maintained by Git in a Bitbucket repository. The repository can be subdivided into subfolders or branches. Developers only have access to files or project versions contained in subfolders or branches that are not used to store the production versions of enhancements. Only administrators have access to the folders that store stage and production versions of enhancements.

The Automations pull the correct state of the file-based enhancements from the associated branches of the Bitbucket repository. This enables the Automations to launch the environments with the appropriate state of the file-based enhancements.

Migrating Data

Among the greatest challenges in migrating an enterprise application is migrating the data. The primary tools that you can use to migrate data for Oracle databases are AWS Database Migration Service (AWS DMS), AWS Snowball (Snowball), Oracle Data Pump (Data Pump) and Oracle Materialized Views (Materialized Views).

AWS DMS is a web service that you can use to migrate data from an on-premises database, an Amazon RDS instance, or a database on an Amazon EC2 instance, to an Amazon RDS database instance or to a database on an Amazon EC2 instance. AWS DMS can also migrate a database from an AWS service to an on-premises database. AWS DMS can migrate data between heterogeneous or homogeneous database engines.[9] In the authors' experience, AWS DMS is an excellent service for smaller databases but might be less reliable for enterprise-class data migrations.

AWS Snowball accelerates transferring large amounts of data into and out of AWS using physical storage devices, which enables you to bypass the internet. Each Snowball device type can transport data at faster-than internet speeds. The devices with the data are shipped through a regional carrier.[10] Because Snowball relies on physical delivery of an appliance to complete the migration, it can create timing uncertainties for migration.

Data Pump provides server-side infrastructure and high-speed, parallel export and import utilities for highly efficient bulk data and metadata movement between databases.[11] Data Pump is a reliable method of migrating data, but does not include delta refresh functionality.

A *Materialized View* is a database object that contains the results of a query. Materialized Views are local copies of data that is located remotely. Materialized Views can query tables, views, and other Materialized Views, and can be used to maintain copies of remote data on a local node.[12] In the authors' experience, a customized Materialized View tool has proven to be a stable and reliable method for migrating an enterprise class database.

Table 1 – Migration tool attribute summary

Migration Tool	Runs to Completion	Continuously Available	Delta Refreshes
AWS DMS		✓	✓
AWS Snowball	✓		
Data Pump	✓	✓	
Materialized Views	✓	✓	✓

Network and Security Design

The architecture of production environments is designed for high availability, among other features. Each layer of the environments—web, application, batch, and database—is spread across two Availability Zones using Amazon ECS, Amazon RDS, or other currently available AWS products and services.

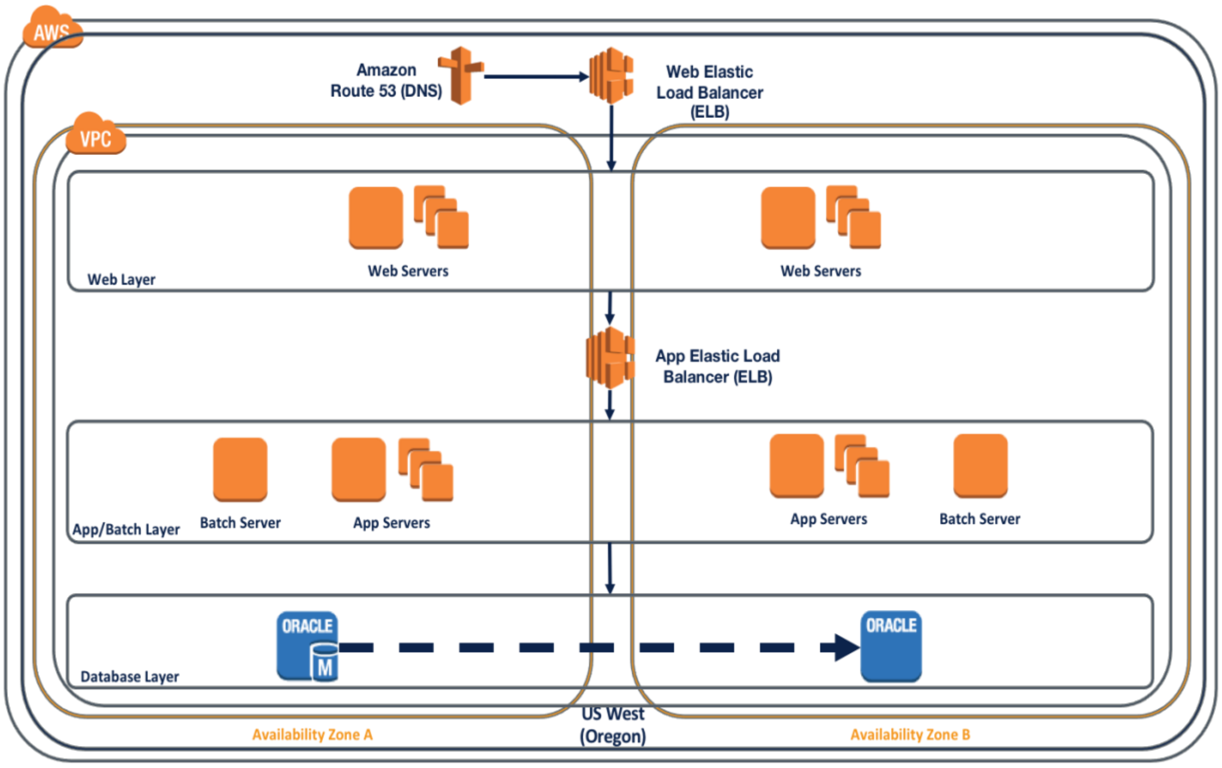


Figure 9 – Example of Availability Zones

Because Amazon ECS functionality allows architects to design tasks that are both launched and monitored automatically, if a task fails, Amazon ECS tries to launch a new task, as defined by the user, in either the original availability zone or the alternate availability zone.

Amazon RDS databases in production environments are configured with high availability (Multi-AZ), which includes automatic failover functionality. This means that if a critical fault in the primary database occurs, the secondary database automatically becomes the primary database. This Multi-AZ availability enables existing AWS network technology to maintain high availability.

AWS features a shared responsibility security structure.[13] Only AWS staff have access to AWS data centers, servers, storage, and other appliances. Only AWS account holders have access to applications and data. All of the AWS services described in this architecture exist in a VPC. Access to the VPC is controlled solely by the AWS account holder. The application, data, and services in the VPC are also controlled exclusively by AWS account holders, based on their security roles, processes, and procedures.

Deploying the Solution

In a traditional PeopleSoft implementation, web, application, batch, and database servers must be set up and configured, one-by-one, manually, environment-by-environment. With this kind of manual process, you are likely to have inconsistencies between environments which become exacerbated over time. With the automated PeopleSoft solution, the Automations build the web, application, batch, and database servers with perfect consistency, because the set up and configuration processes share a common, underlying code. If servers become corrupt over time, they can be relaunched with very little effort to restore consistency and functionality.

Managing the Solution

Some aspects of managing the automated PeopleSoft solution are identical to managing an on-premises solution, but others are radically different.

1. **Scaling** – AWS Auto Scaling functionality, linked to such metrics as CPU or memory utilization, can accommodate small changes in traffic without manual intervention.[14] Event scaling, such as that required for university registration or year-end reporting, is accomplished using the Automations.
2. **Launching and terminating** – Because the Automations can launch the non-production environments in about 40 minutes, the non-production environments can be terminated during non-business hours and relaunched immediately prior to the start of business hours.
3. **Application patches and upgrades** – Any upgrade that alters the state of the database must use traditional methods to make changes to the database. This includes using an upgrade tool and reapplying customizations, instance-by-instance, to the upgraded database.
4. **Server upgrades** – To upgrade your server, revise your Dockerfiles to show the current version of the server software. Create a new image with Docker Engine and tag it as the new version. Then use Amazon ECS to call the new Docker image and launch the container services. When the new image is stored in Amazon ECR, the new version of the servers can be launched repeatedly for various pillars, or servers in a pillar.
5. **Database upgrades** – Because the Automations use Amazon RDS to provide the database services, you can upgrade your database by simply calling the new version of Amazon RDS database; no other steps are required. For RDS instances with Multi A-Z enabled, the upgrade occurs without an outage. If Multi A-Z is not enabled, a brief outage occurs.

Conclusion

AWS provides a tremendous amount of on-demand infrastructure services for cloud storage. Until recently, PeopleSoft users who migrated to AWS have used the many AWS infrastructure services with a data center outsourcing model. Though this model uses AWS infrastructure services, the implementation is subject to the same pillar-by-pillar inconsistencies and degradation over time as an on-premises implementation.

The migration approach recommended in this paper augments delivered AWS infrastructure services functionality with the Automations, which empower users to much more effectively and economically administer a PeopleSoft implementation. Because the Automations can launch the non-production instances so quickly, these instances can be terminated during non-business hours. Because the Automations are used to administer all the environments in the implementation, a high degree of consistency is maintained among the environments. The stability of all the environments is also engineered into the Automations. Improvements are managed by re-engineering the Automations and are then quickly pushed out to all the environments after they are validated and approved.

PeopleSoft administrators and users can get great administrative and economic benefits from using AWS services as on-demand infrastructure services.

Contributors

Authors of this document include:

- Jeff Davis, Partner, The Burgundy Group, Inc.
- Josh Shaloo, Managing Director, The Burgundy Group, Inc.

Additional contributors to this document include:

- Ashok Shanmuga Sundaram, Solutions Architect, Amazon Web Services

Document Revisions

Date	Description
April 2019	First publication

Notes

[1] Overview of Amazon Web Services
http://d0.awsstatic.com/whitepapers/aws-overview.pdf

[2] Docker Overview
https://docs.docker.com/engine/docker-overview/

3 Images and Containers
https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/containers_and_images.html

4 Docker Image Specification v1.0.0
https://github.com/moby/moby/blob/master/image/spec/v1.md

5 Digging Into Docker Layers
https://medium.com/@jessgreb01/digging-into-docker-layers-c22f948ed612

6 What is CloudFormation?
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

7 Amazon Elastic Container Service
https://aws.amazon.com/ecs/

8 What is Amazon Relational Database Service (Amazon RDS)?
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

9 AWS Database Migration Service Documentation
https://docs.aws.amazon.com/dms/index.html#lang/en_us

10 AWS Snowball User Guide
https://docs.aws.amazon.com/snowball/latest/ug/whatissnowball.html

11 Oracle Data Pump Overview
https://www.oracle.com/technetwork/documentation/data-pump-overview-084963.html

12 Materialized Views In Oracle
https://www.databasejournal.com/features/oracle/article.php/2192071/Materialized-Views-in-Oracle.htm

13 AWS Security Best Practices
https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf

14 AWS Auto Scaling
https://aws.amazon.com/autoscaling/