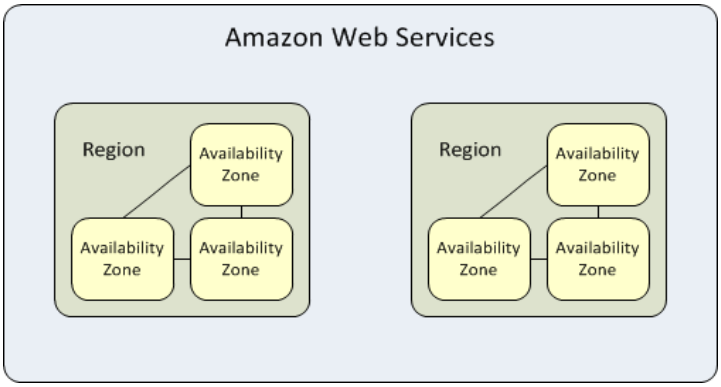
Chapter 13 – Clouducate

While we hope that this book provides a solid understanding of how Enterprise IT environments work and provides the tools for troubleshooting issues within such environments, there's nothing like experiencing problems firsthand. For this reason, the authors created a set of tools, collectively called Clouducate, that create a virtual data center in the cloud with a couple of web-based applications. These environments are then configured with various errors from assignment to assignment to allow students to utilize all the tools covered in this book to try to figure out the problems.

At this time, these tools require the use of AWS and a small amount of spend on AWS services (about $40-$50 per semester if care is taken to stop services when not using them) unless education credits can be acquired from AWS.

13.1 AWS Glossary and Concepts

Before we get into how Clouducate works, we thought it would be best to review AWS terms and concepts first as understanding at least some AWS basics is a key part of being able to troubleshoot the issues created by Clouducate.

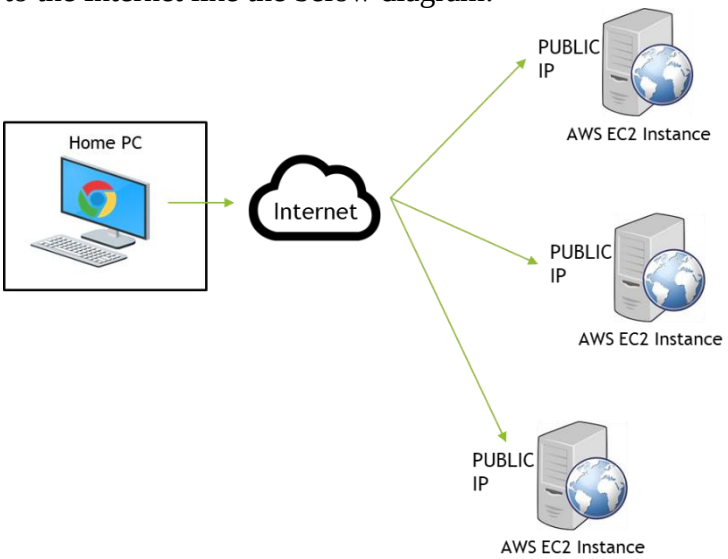


- AWS Region – One or more AWS data centers within a nearby area (within 250 miles) – We will be using "us-east-2"
- AWS Availability zone – One datacenter within a region - We will be using 1 AZ in this course – "us-east-2A"
- AWS VPC (Virtual Private Cloud) – Private network within AWS includes large private address space; can span Availability zones, but not regions
  - We will be using this as it simulates an Enterprise environment
- AWS VPC Subnet – Portion of VPC addresses used to group instances & services; Cannot span availability zones
- IAM User – Need to create a user (any name) to administrate your services as opposed to the account login
- User Access Key ID and Access Secret Key (associated to IAM user) – Needed for AWS CLI (command-line interface) scripts
- S3 Storage – AWS' Simple Storage Service (object storage via Internet access)
  - We will be using this service for Load Balancer logs
- EC2 (Elastic Compute Cloud) Instance – A VM server in AWS (can be associated with a VPC subnet and a security group)

- ec2-user – Userid to login to your Linux EC2 instances
- Administrator – Userid to login to your Windows instances
- EC2 instance states – running, stopped, and terminated (deleted)
- Routing table – Associated to VPC subnets – how to route network requests
- Security Group – Associated to EC2 instances/AWS services – firewall rules for access in & out of instance/service
- NAT Instance – A pre-configured EC2 instance that allows outbound access to the Internet
- Internet Gateway - Allows Internet access out of VPC
- Client VPN Endpoint – Allows VPN access into VPC - Associated to a VPC subnet
- Route 53 – AWS' DNS service - Associated to VPC IP addresses (inbound endpoints)
  - Can include private zone (e.g. AWSVPCB.edu) for DNS resolution within VPC
- ELB (Elastic Load Balancer) – AWS' load balancer service (there are multiple options – we will utilize the Classic LB) - Associated to subnet & security group

13.2 Clouducate Environment and AWSVPCB Scripts

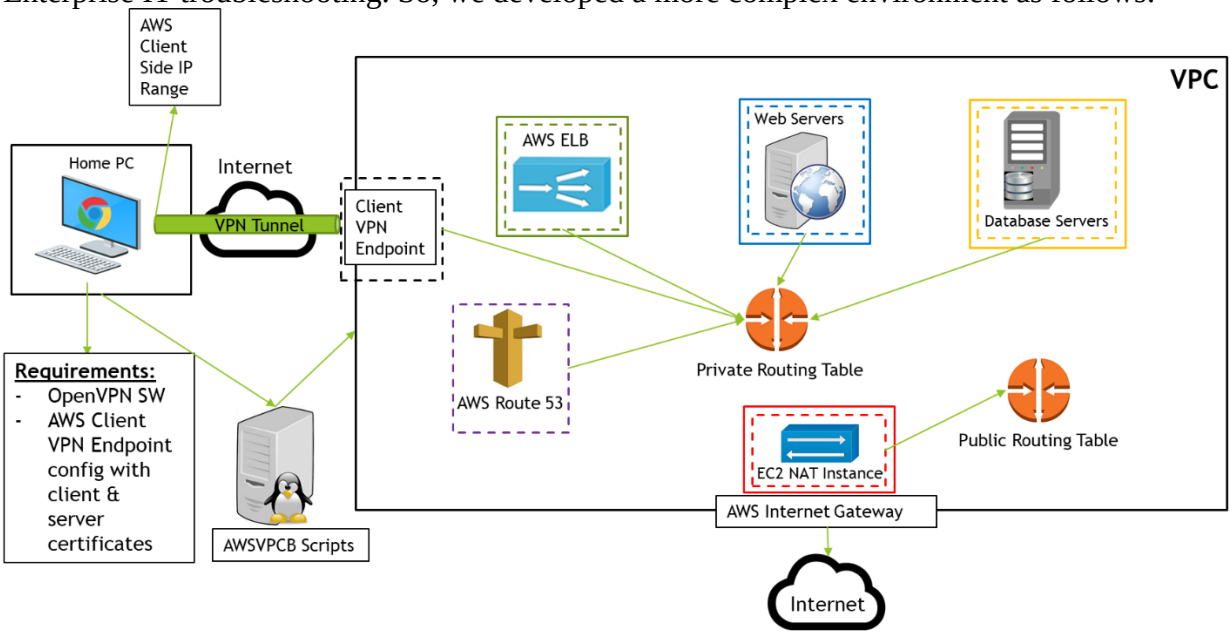
As we began to work on Clouducate, we wanted to keep the architecture simple, but not too simple. For example, to one extreme, we could create servers in AWS and simply expose them to the Internet like the below diagram:



However, this would be completely irrelevant to what an Enterprise IT environment looks like. Just some of the problems with a setup like this would be as follows:

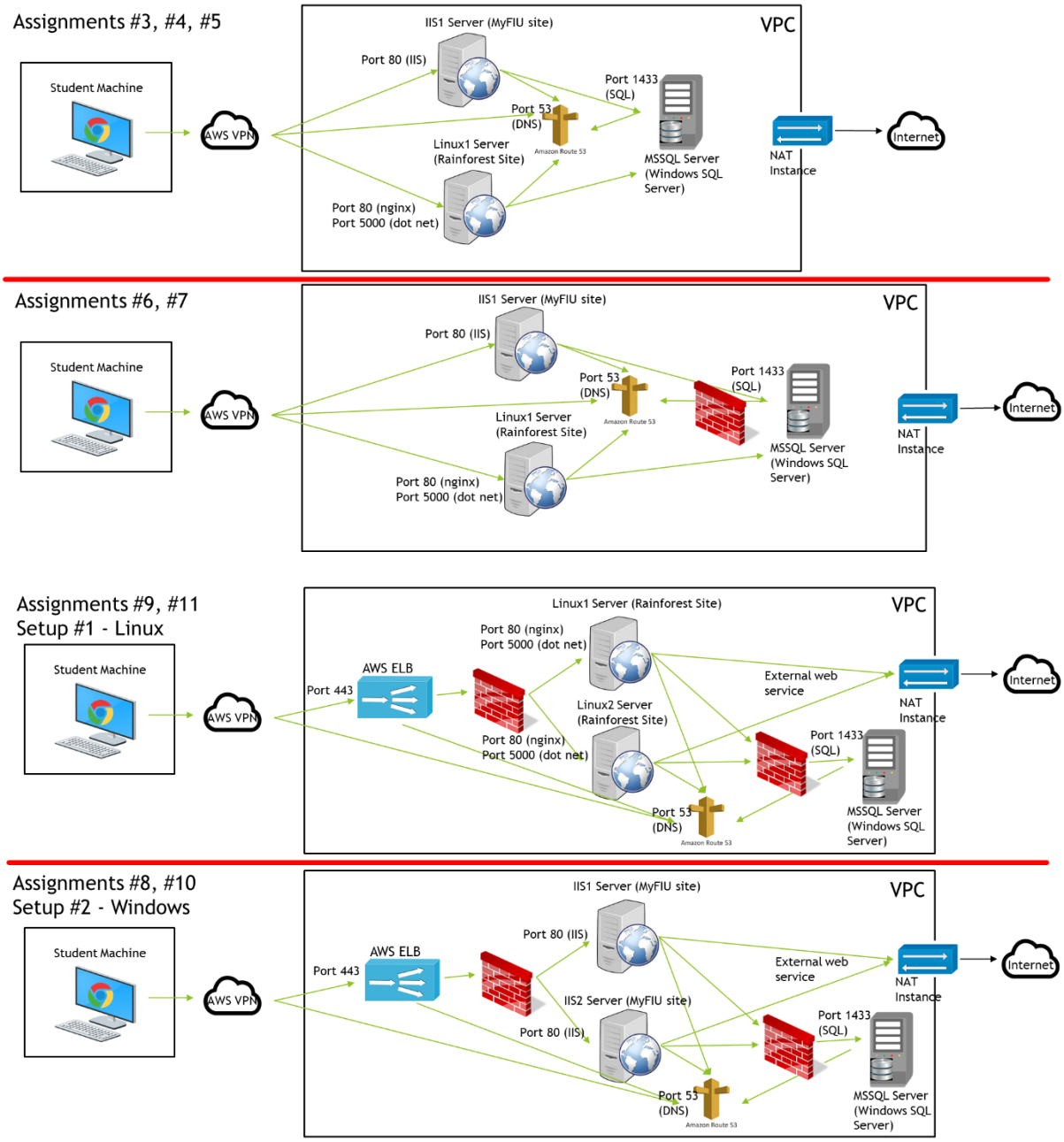
- All servers are exposed to the Internet (no private addressing)
- No local DNS servers
- No firewall zones
- No load balancing
- No subnetting
- No internal routing
- No VPN

In general, this would be unrealistic in terms of simulating any kind of significant real-world Enterprise IT troubleshooting. So, we developed a more complex environment as follows:



This setup includes six subnets (dotted lined boxes), four of these subnets are protected by a security group (solid lined boxes) limiting their access in and out of the subnet to simulate firewalls, all private addressing except for the exit point (AWS Internet Gateway), VPN connectivity, internal DNS, internal routing, and finally applications with load balancers, web/application servers and a database server. Also noted is a Linux server to execute the AWSVPCB scripts – this Linux server is used to create the VPC and its components but does not have access into the VPC. NOTE: a MacOS can be used for the scripts as well.

While the complexity of this setup is still minor compared to real-world Enterprise IT environments, it at least provides enough to allow for students to learn how to use some of the tools that are used in real enterprise environments. In fact, there are four distinct environment setups across nine assignments. These environments increase in complexity over the course as the students learn more concepts. The below are the environment diagrams for the assignments:



Each student creates their own environment to allow each of them to work independently, although the course also calls for group assignments. In order for a student to create their own AWS environment, they first need to find a Linux or MacOS machine onto which they can install the AWS CLI (version 2) as well as the Clouducate AWSVPCB shell scripts and supporting files. The Clouducate AWSVPCB scripts do not support Windows at this time.

The below is a description of the AWSVPCB scripts and what they do:

- 5 one-time scripts as follows and 1 recovery script:

- AWSVPCB.CONFIGURE & AWSVPCB.TEST – Only needed at the start of the course, but can be run again, if needed.
- AWSVPCB.VPC.CREATE – This script creates your VPC, Internet Gateway, route tables, subnets, security groups, and Client VPN endpoint. It also registers all the AWS unique IDs generated during the builds
  - This script will fail if an "AWS-VPCB" tagged VPC already exists
  - You should only need to run this script once. If you need to start from scratch, you should run AWSVPCB.VPC.DESTROY before re-running this.
- AWSVPCB.VPC.DESTROY – This script will destroy the registered VPC and everything in it.
  - You should only need to run this at the end of the course; running this will require replacing the VPN config.
- AWSVPCB.VPC.REGISTER – This script may never be needed. It will find your AWS-VPCB VPC in AWS and register all its components for the other scripts to be able to work as expected.

▶ 4 multi-use assignment scripts as follows:

- AWSVPCB.ASSIGNMENT.CREATE # – Destroys existing instances and ELB targets if any exist, then Adjusts VPC settings, Creates assignment instances and ELB (if applicable) based on number passed in as parameter; **Destroys any existing work you've done on assignment.**
- AWSVPCB.ASSIGNMENT.START – Starts instances, Associates Client VPN endpoint to subnet, Creates Route 53 DNS Zone, DNS entries (including saved entries if the assignment was previously stopped) and DNS Inbound endpoints; Can be run multiple times without destroying work.
- AWSVPCB.ASSIGNMENT.STOP – Stops instances, Disassociates Client VPN endpoint from the subnet, Saves DNS entries, Deletes Route 53 DNS Zone and Inbound Endpoints; Can be run multiple times without destroying work.
- AWSVPCB.ASSIGNMENT.DESTROY – Destroys existing ELB and instances. Called by AWSVPCB.ASSIGNMENT.CREATE; **Destroys all the work you've done on the assignment.**

In the course taught by the authors, $100 in AWS credits are provided to each student. However, care must be taken by the students to avoid overrunning these credits. The below is guidance provided in terms of the usage of these credits.

Credit Usage Implications of Scripts: The below assumes a new AWS account. If you have an account older than 12 months will be charged a small amount after VPC.CREATE and a little more after ASSIGNMENT.CREATE. If you are in this position, be more mindful of your spend status, but should still have enough credits for the course.

- AWSVPCB.VPC.CREATE - No charges after this script is run.
- AWSVPCB.ASSIGNMENT.CREATE - After running this script for later assignments that use the ELB, a few cents will begin to be billed to your account daily.
- AWSVPCB.ASSIGNMENT.START - You will not be able to use your services until you run this script. After you run this script, you will begin to draw down on your credits at a much steeper rate (several dollars per day). The course is designed to allow you hundreds of hours of uptime, but not 24x7 for days. **If you do not run the**

AWSVPCB.ASSIGNMENT.STOP script, whenever you are done working on your assignment, you will run out of credits.

- AWSVPCB.ASSIGNMENT.STOP - will stop the necessary services so that you are no longer using a heavy amount of credits while saving all of your changes. While your changes are saved, you will not be able to work until you perform an ASSIGNMENT.START again.
- AWSVPCB.VPC.DESTROY - If you find that you need to destroy your VPC for a while to save on credits by running this script, that is fine, but please note that you will need to re-import your VPN configuration after recreating your VPC.

13.3 Clouducate Components and Interdependencies

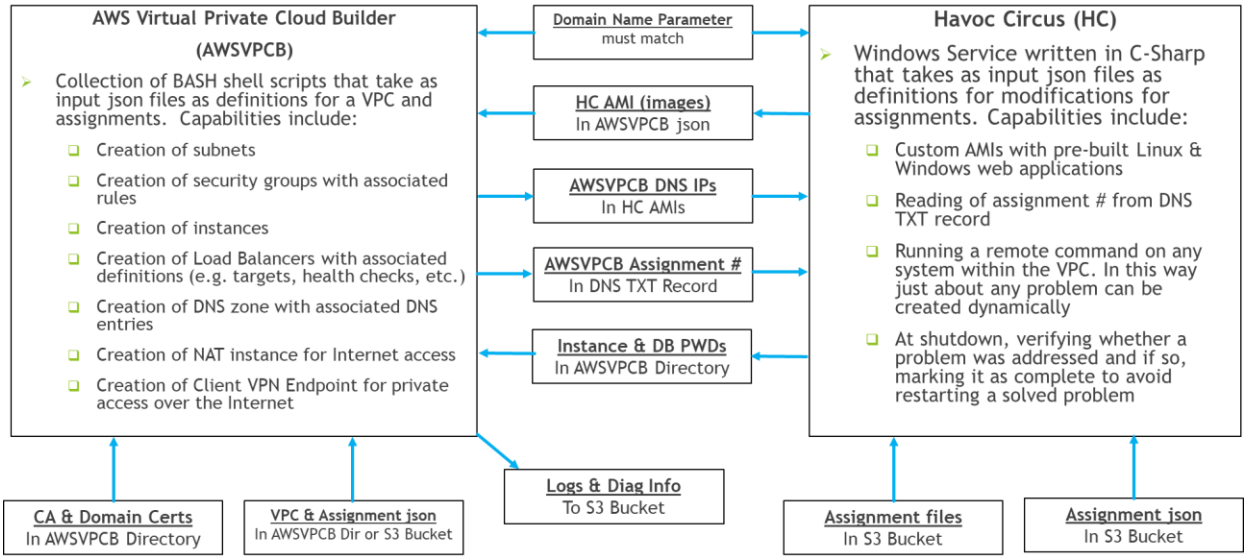
At a high level, Clouducate is currently made up of two independent codebases that coordinate with each other to create a dynamic, highly flexible environment. These two components are AWSVPCB scripts, which were discussed in the previous section because that's what the students are exposed to, and Havoc Circus, which allows for the dynamic alteration of the environment to create problems for the students to solve. Both of these components utilize JSON configuration files that are retrieved from the cloud to build each assignment uniquely.

There are several intersection points between these components such as the domain name for the environment (i.e. the DNS zone – defaults to awsvpcb.edu), the AMIs (AWS Machine Images) which host the applications/databases, DNS servers which need to be configured in the AMIs, Assignment number which is passed between these components through a DNS TXT record and finally the passwords for the instances and databases which are included in the AWSVPCB support files for the students.

The AWSVPCB scripts are written as Bash shell scripts. These scripts create the AWS VPC and all of its components as described in the previous section. These scripts are accompanied by several critical files including the passwords for all the systems, SSL Certificate Authority certificate that needs to be trusted for VPN & SSL to the web sites to work, SSL certificate for the VPN and the websites, and the JSON files needed for each assignment.

Havoc Circus is a Windows service written in C-Sharp. The AMI Havoc Circus executes on needs to have SSH tokens for any Linux server it needs to modify as well as access to a SQL Server database where it maintains its state. As such, Havoc Circus is typically installed on the SQL Server that is used to support all the applications in all the assignments. Havoc Circus also retrieves the JSON assignment settings from the cloud at execution time.

All of the above is graphically displayed in the below diagram:



13.4 AWSVPCB Documentation

AWSVPCB (AWS Virtual Private Cloud Builder) is a set of BASH Shell scripts designed to create a small Enterprise IT-like environment in AWS to provide college-level students with hands-on experience individually working on IT problems. These scripts work in conjunction with the Havoc Circus utility which provides AWS images (AMIs) with pre-loaded applications and a method for automatically changing the environment for assignment purposes. These scripts were originally developed by Norbert Monfort and Robert Fortunato for the CTS-4743 Enterprise IT Troubleshooting course taught at FIU (Florida International University) in Miami, Florida. However, these scripts and the associated Havoc Circus C#.Net packages are open-source and free to be used for any purpose. These two components make up what is called the "Clouducate Suite", which we would like to see expand to include additional tools in the future. Both of these components are made generally available as open source. The rest of this section will focus on AWSVPCB.

STRUCTURE OF AWSVPCB

The AWSVPCB scripts require a specific directory structure. You can simply download the awsvpcb-scripts.zip file to install this directory structure to get started. The following is an explanation of the directories:

1. The root directory of the scripts - All the scripts that are meant to be executed by the students reside in this directory and are all fully CAPITALIZED. All the other directories are support directories relevant to the instructor, but not to the students. The root directory also includes the vpcb-config file which provides for a few configuration settings for the scripts.
2. The "procs" directory includes all of the detail code for the scripts as well as some dynamic variable files (collectively called the registry) that start empty but are modified as the scripts load the JSON files and build out the AWS components. All "code" resides in this directory and the root directory.
3. The "secfiles" directory includes all the supporting information like:
    - The certificates and private keys used by the load balancers

- The certificates and private leys used by the VPN (client & server-side)
- The private key used for the AWS instances
- The certificate authority certificate for all the above
- The passwords for all of the instances so that the students can log in
- The password for the database so that students can log in
- The dynamically generated OVPN file for the students to import into their PC for VPN connectivity
- The dynamic JSON files and templates used to create and save DNS entries
- The downloaded VPC and Assignment-level JSON files used to build the VPC(s) and assignments (these are in sub-directories)

4. The "tempfiles" directory includes temporary dynamically generated files
5. The "logs" directory includes all of the output for all the scripts run

CONFIG FILE (vpcb-config) SETTINGS

There aren't many settings to worry about in the vpcb-config file and most are self-explanatory, but here's the list:

- COURSE & SEMESTER - These two parameters go hand in hand to determine the name of the log group to be created in Cloudwatch. If you happen to have multiple sections for a particular course within a semester, then it's recommended that you add the section number to the COURSE parameter.
- DOMAIN - This is used to append to all of your server and ELB names. Consider this your fake companies domain name. The default is "awsvpcb.edu" and certificates for the VPN, and a couple of ELB (load balancer) names are provided for this domain as well as the CA cert (ca.crt) which is all used to build the VPC and ELBs. However, if you wish to create new ELB names or use a different domain, then you would need to create your own CA and replace all the relevant files in the "secfiles" directory. Changing the domain also implies that you have created your own AMIs Havoc Circus assignment files compatible with the new domain name.
- AWSCMD - This parameter should not be changed for now. It was set up to provide for the ability to change the aws command in the future.
- ENABLE_AWS_LOGGING - This parameter can be set to "yes" or "no". If set to "yes", then an AWS access key and secret key with access to your Cloudwatch logs must be provided.
- MANIFEST_LOCATION - This parameter can be set to "aws" or "local". If set to "aws", then an AWS access key and secret key with access to your S3 bucket where the VPC and Assignment JSON files are located.
- AWS_REGION - The AWS region where you want your VPC built. This must be the same region that you have instructed your students to select when running the “aws configure” command after the AWS CLI installation.
- AWS_AZ - The AWS availability zone where you want your VPC built.
- LOGGING_ACCESS_KEY - The AWS access key to be used for logging into AWS.

- LOGGING_SECRET_KEY - The AWS secret key to be used for logging into AWS.
- MANIFEST_ACCESS_KEY - The AWS access key to be used to download VPC and assignment manifest JSON files from AWS.
- MANIFEST_SECRET_KEY - The AWS secret key to be used to download VPC and assignment manifest JSON files from AWS.
- MANIFEST_S3BUCKET - The AWS S3Bucket where the VPC and assignment manifest JSON files are located in AWS.
- DIAGLOG_S3BUCKET - The AWS S3Bucket where diagnostic information will be placed; the LOGGING_ACCESS_KEY must have access to write to this S3Bucket.

MAIN EXECUTABLE SCRIPTS

- AWSVPCB.CONFIGURE - Reads vpcb-config and configures the base settings within the "procs" directory. Can be rerun as often as needed but does not need to be run unless a change is made to the vpcb-config file.
- AWSVPCB.TEST – Simply tests basic connectivity to AWS.
- AWSVPCB.VPC.CREATE – This script optionally accepts a numeric parameter (the VPC number; 0 is the default if no number is provided). The script expects a vpc# directory (where # is the VPC number) to exist in the "secfiles" directory or AWS manifest S3bucket where the vpc.json file can be found and loaded. Using this vpc.json file, this script creates a VPC with associate Internet Gateway, NAT instance, route tables, subnets, security groups, S3Bucket for ELB logs, Client VPN endpoint, OVPN file for the OpenVPN client and registers all AWS unique IDs. This script will fail if an “AWS-VPCB” tagged VPC already exists in the target AWS account. To rerun this script, the AWSVPCB.VPC.DESTROY must be run first.
- AWSVPCB.VPC.DESTROY – This script will destroy the registered VPC and everything in it.
- AWSVPCB.VPC.REGISTER – This script compares the AWS object IDs registered with what exists in AWS to adjust the registry files in the "procs" directory appropriately. The script requests the user to provide an optional assignment number such that all of the assignment components are also registered properly.
- AWSVPCB.ASSIGNMENT.CREATE # – This script accepts a mandatory assignment number (no default). The script then stops and destroys existing assignment instances, DNS entries, and ELB targets if any exist. The script then expects an assignment# directory (where # is the assignment number) to exist in the "secfiles" directory or AWS manifest S3bucket where the awsvpcb.assignment#.json can be found and loaded. The script then creates assignment instances and firewall rules.
- AWSVPCB.ASSIGNMENT.START – Starts instances, Creates ELB (if applicable and just during the first start), Associates Client VPN endpoint to subnet, Creates Route 53 DNS zone and entries, Creates Route 53 Inbound endpoints. This script can run multiple times without destroying any work.
- AWSVPCB.ASSIGNMENT.STOP – Stops instances, Saves Route 53 DNS entries, Destroys Route 53 DNS zone, Disassociate Client VPN endpoint from the subnet,

Deletes Route 53 Inbound Endpoints. This script can run multiple times without destroying work.

- AWSVPCB.ASSIGNMENT.DESTROY – Destroys existing assignment ELB and instances. This script is called by AWSVPCB.ASSIGNMENT.CREATE and destroys all the work done on the assignment.
- AWSVPCB.DIAGLOG - This script gathers all the relevant information from the student's AWSVPCB directories and AWS VPC and sends it to an S3 bucket for review. All the files are placed in the S3Bucket defined in the vpcb-config file.
- AWSVPCB.MANIFEST.DISPLAY - This script prompts the user for which manifest to display (VPC or Assignment and the relevant #). The script then reads the designated manifest and displays what was extracted and would be loaded into the registry of the awsvpcb-scripts. This is a good way to test whether your JSON is properly formatted and accepted. Note that this does not validate the JSON, just displays what loading it would do.

VPC JSON FILES

The awsvpcb-scripts.zip file includes a sample JSON configuration file in the "secfiles/vpc0" directory. AWSVPCB allows for the definition of multiple VPCs (default is vpc0), however, there could only be one defined in AWS for a given AWS account at one time. Also, because each time you create a VPC a new OVPN file is generated, it is recommended that you limit the number of VPCs used in one course. For CTS-4743, for example, we only use one VPC for the entire semester and use the assignment JSON files to adjust the environment. VPCs can be created, registered, and destroyed. Registering a VPC entails comparing the dynamic files within the "procs" directory with what is actually in AWS. All VPC JSON parameters are required, albeit the number of subnets, possibleInstanceNames, and PossibleELBs is variable. The sample VPC JSON file includes all the parameters currently available. The below is a summary of these parameters:

- VPC-VPCCIDR(required): The range of IPs available for the VPC in CIDR notation
- Subnets(required): The number of subnets is variable, but the DEFAULT and PUBLIC subnets are required and should not be touched
- Subnets-SubnetName(required): The name of the subnet (no spaces allowed)
- Subnets-SubnetCIDR(required): The IP range for the subnet in CIDR notation
- Subnets-SecurityGroup(required): "yes" or "no" as to whether this subnet has an equivalently named Security group controlling its inbound and outbound traffic
- Subnets-RoutingTable(required): "DEFAULT" or "PUBLIC" - all subnets other than the PUBLIC, should use the DEFAULT routing table
- PossibleInstanceNames(required): The number of PossibleInstanceNames is variable, but at least one must exist. This is necessary to allow the AWSVPCB.VPC.REGISTER script to re-calibrate the registry for the scripts with what exists in AWS. Simply list the possible instance names that may exist in any assignment to be used with this VPC.
- PossibleELBs(required): The number of PossibleELBs is variable, but at least one must exist. This is necessary to allow the AWSVPCB.VPC.REGISTER script to re-calibrate the registry for the scripts with what exists in AWS. Simply list the possible ELB names that may exist in any assignment to be used with this VPC.

- DNSIPAddresses(required): This is a list of the IP addresses that will be defined in the AWS Route 53 resolver (DNS server). All AMIs should have these IPs in their config to appropriately resolve your private domain's DNS names to IPs.
- NATDefinition(required): This is unlikely to need to be changed as this is simply defined to allow Internet access from within the VPC.
- NATDefinition-IPAddress(required): IP address for the NAT instance.
- NATDefinition-Subnet(required): Subnet for the NAT instance.
- NATDefinition-AMI(required): AMI (AWS Machine Image) for the NAT instance.

SAMPLE AWSVPCB VPC JSON INCLUDED IN ZIP FILE (secfiles/vpc0/vpc.json)


```
{
 "AWSVPCB": {
  "VPC":
    {
     "VPCCIDR": "172.31.0.0/16"
    }
 },
 {
  "Subnets": [
    {
     "SubnetName": "DEFAULT",
     "SubnetCIDR": "172.31.131.0/24",
     "SecurityGroup": "yes",
     "RoutingTable": "DEFAULT"
    },
    {
     "SubnetName": "WEB",
     "SubnetCIDR": "172.31.128.0/24",
     "SecurityGroup": "yes",
     "RoutingTable": "DEFAULT"
    },
    {
     "SubnetName": "DB",
     "SubnetCIDR": "172.31.129.0/24",
     "SecurityGroup": "yes",
     "RoutingTable": "DEFAULT"
    },
    {
     "SubnetName": "ELB",
     "SubnetCIDR": "172.31.130.0/24",
     "SecurityGroup": "yes",
     "RoutingTable": "DEFAULT"
    },
    {
     "SubnetName": "CLIENTVPN",
     "SubnetCIDR": "172.31.16.0/24",
```

```
      "SecurityGroup": "no",
      "RoutingTable": "DEFAULT"
    },
    {
      "SubnetName": "PUBLIC",
      "SubnetCIDR": "172.31.132.0/24",
      "SecurityGroup": "yes",
      "RoutingTable": "PUBLIC"
    }
  ]
},
{
  "PossibleInstanceNames": [
    {
      "InstanceName": "IIS1"
    },
    {
      "InstanceName": "IIS2"
    },
    {
      "InstanceName": "LINUX1"
    },
    {
      "InstanceName": "LINUX2"
    },
    {
      "InstanceName": "MSSQL"
    }
  ]
},
{
  "PossibleELBs": [
    {
      "ELBName": "myfiu"
    },
    {
      "ELBName": "rainforest"
    }
  ]
},
{
  "DNSIPAddresses": [
    {
      "IPAddress": "172.31.131.10"
    },
    {
```

```
      "IPAddress": "172.31.131.11"
     }
    ]
  },
  {
   "NATDefinition":
     {
      "IPAddress": "172.31.132.151",
      "Subnet": "PUBLIC",
      "AMI": "ami-****"
     }
  },
  {
   "VPNDefinition":
     {
      "CACert": "ca.crt",
      "ConfigFile": "AWSVPCB-client-config.ovpn",
      "ClientCIDR": "172.31.8.0/22"
     }
  },
  {
   "ServerPrivateKey": "privkey"
  }
}
```


ASSIGNMENT MANIFEST JSON FILES

The awsvpcb-scripts.zip file includes sample assignment JSON configuration files in the "secfiles/assignment#" directories (#=1-3). AWSVPCB allows for the definition of multiple assignments (there is no default). Assignments can be created, started, stopped, and destroyed. Starting and stopping an assignment preserves all changes made. The option to start and stop an assignment is necessary to allow a student to step away and not unnecessarily use up their allotted AWS credits. The three sample assignment JSON files include all the parameters currently available. Some of these parameters are optional. The below is a summary of these parameters:

- Instances(required): The number of instances is variable, but at least one must exist
- Instances-InstanceName(required): The name of the instance (no spaces allowed)
- Instances-InstanceIP(required): The IP address for this instance. Must be within the IP range of the subnet
- Instances-InstanceSubnet(required): The subnet for the instance. Must be a subnet defined in the VPC.json file
- Instances-InstanceAMI(required): The AMI (AWS Machine Image) for this instance. If the AMI is private, then it must be shared with the student's AWS account
- Instances-InstanceType(required): The type/size of the instance. While anything can be chosen here, please be aware that only type t2.micro is currently covered by the AWS "free tier", which allows for many more hours of usage without chewing up a lot of AWS credits.

- Instances-StartPreference(optional): Although this can be used for any purpose, it is only necessary for the server that executes the Havoc Circus service and should be set to "first" ("last" is also an available option, but will cause problems if used for the instance running the Havoc Circus service)
- Instances-StopPreference(optional): Although this can be used for any purpose, it is only necessary for the server that executes the Havoc Circus service and should be set to "last" ("first" is also an available option, but will cause problems if used for the instance running the Havoc Circus service)
- FirewallRules(optional): Syntactically, Firewall Rules do not need to be provided and there is no limit as to how many are provided. However, if rules are not provided, then no access will be allowed into a security group, so generally, at least one inbound and one outbound rule per security group is needed to make things functional.
- FirewallRules-SecurityGroup(required): The security group to which this particular firewall rule applies
- FirewallRules-RuleType(required): Must be "inbound" or "outbound"
- FirewallRules-Protocol(required): Can be "all", "udp", "tcp" or "icmp"
- FirewallRules-Port(required): Can be "all" or specific port number or port range with a hyphen in between (e.g., "137-139")
- FirewallRules-SourceGroup(required): The source (for "inbound" rules) or destination (for "outbound" rules) for this firewall rule in CIDR notation
- VPNFlag(optional): The default is “enable”. Set this to “disable” to avoid creating a ClientVPN endpoint and to use a Bastion host instead to access your VPC over the Internet. Using a Bastion host carries a bit more cost but does avoid complexity and VPN issues.
- DNSEntriesFile(required): This is the location of the JSON file that includes the DNS entries to apply to Route 53 when the assignment is created. The path is relative to the "secfiles" directory and/or the AWS S3 bucket where the manifest exists.
- ELBs(optional): Only required if ELBs are defined. NOTE: There is a log automatically created for each ELB within an S3Bucket within the student's AWS account that will include all the requests into that ELB (equivalent to a web access log). There is also a DNS entry automatically created for the ELB.
- ELBs-ELBName(required): The name of the ELB. There should be an SSL certificate (.crt file) and private key (.pkey file) in the "secfiles" directory with this name and the chosen domain appended (e.g. rainforest.awsvpcb.edu.crt and rainforest.awsvpcb.edu.pkey where rainforest is the ELBName and awsvpcb.edu is the DOMAIN).
- ELBs-ListenerProtocol(optional): The default is HTTPS. At this time, no other option is supported, although support for HTTP is being built to avoid the need for certificates.
- ELBs-ListenerPort(optional): The default is 443. Any value recognized by AWS is accepted.
- ELBs-InstanceProtocol(optional): The default is HTTP. Any value recognized by AWS is accepted.
- ELBs-InstancePort(optional): The default is 80. Any value recognized by AWS is accepted.

- ELBs-ELBSubnet(optional): The default is ELB. The name chosen must be a valid subnet as defined in the VPC JSON file
- ELBs-HealthCheckTarget(optional): The default is TCP:80. This will be what AWS uses to validate that the target instances are active. Any value recognized by AWS is accepted.
- ELBs-HealthCheckInterval(optional): The default is 5. This indicates how often AWS will execute the Health Check (in secs). Any value recognized by AWS is accepted.
- ELBs-HealthCheckTimeout(optional): The default is 3. This indicates how long (in secs) before AWS considers a health check as timed out (failed). Any value recognized by AWS is accepted.
- ELBs-HealthCheckUnhealthyThreshold(optional): The default is 2. This indicates how many health checks must fail for the target instance to be taken out offline. Any value recognized by AWS is accepted.
- ELBs-HealthCheckHealthyThreshold(optional): The default is 2. This indicates how many health checks must succeed for the target instance to be brought back online. Any value recognized by AWS is accepted.
- ELBs-EnableSessionStickiness(optional): The default is N for "no". This indicates whether the load balancer should enable cookie-based session stickiness such that sessions remain on the same backend server. N for "no" or Y for "yes" are accepted.
- ELBs-ELBInstances(required): The number of instances is variable, but at least one instance is required.
- ELBs-ELBInstances-InstanceName(required): The name of a target instance for the ELB. Must be an instance defined in the assignment.

ONE OF THE THREE SAMPLE AWSVPCB ASSIGNMENT JSON INCLUDED IN ZIP FILE (secfiles/assignment1/awsvpcb.assignement1.json)


```
{
 "AWSVPCB": {
  "Instances": [
   {
    "InstanceName": "LINUX1",
    "InstanceIP": "172.31.128.43",
    "InstanceSubnet": "WEB",
    "InstanceAMI": "ami-****",
    "InstanceType": "t2.micro"
   },
   {
    "InstanceName": "LINUX2",
    "InstanceIP": "172.31.128.44",
    "InstanceSubnet": "WEB",
    "InstanceAMI": "ami-****",
    "InstanceType": "t2.micro"
   },
   {
    "InstanceName": "MSSQL",
    "InstanceIP": "172.31.129.75",
```

```
      "InstanceSubnet": "DB",
      "InstanceAMI": "ami-****",
      "StartPreference": "last",
      "StopPreference": "first",
      "InstanceType": "t3.small"
    }
  ]
},
{
  "FirewallRules": [
    {
      "SecurityGroup": "DEFAULT",
      "RuleType": "outbound",
      "Protocol": "all",
      "Port": "all",
      "SourceGroup": "0.0.0.0/0"
    },
    {
      "SecurityGroup": "PUBLIC",
      "RuleType": "outbound",
      "Protocol": "all",
      "Port": "all",
      "SourceGroup": "0.0.0.0/0"
    },
    {
      "SecurityGroup": "WEB",
      "RuleType": "outbound",
      "Protocol": "all",
      "Port": "all",
      "SourceGroup": "0.0.0.0/0"
    },
    {
      "SecurityGroup": "DB",
      "RuleType": "outbound",
      "Protocol": "all",
      "Port": "all",
      "SourceGroup": "0.0.0.0/0"
    },
    {
      "SecurityGroup": "ELB",
      "RuleType": "outbound",
      "Protocol": "all",
      "Port": "all",
      "SourceGroup": "0.0.0.0/0"
    },
    {
```

```
 "SecurityGroup": "DEFAULT",
 "RuleType": "inbound",
 "Protocol": "all",
 "Port": "all",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "PUBLIC",
 "RuleType": "inbound",
 "Protocol": "all",
 "Port": "all",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "udp",
 "Port": "137-138",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "80",
 "SourceGroup": "172.31.16.0/24"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "80",
 "SourceGroup": "172.31.130.0/24"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "135-139",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "49152-65535",
```

```
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "3389",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "445",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "22",
 "SourceGroup": "172.31.0.0/16"
},
{
 "SecurityGroup": "WEB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "5000",
 "SourceGroup": "172.31.16.0/24"
},
{
 "SecurityGroup": "DB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "1433",
 "SourceGroup": "172.31.16.0/24"
},
{
 "SecurityGroup": "DB",
 "RuleType": "inbound",
 "Protocol": "tcp",
 "Port": "1433",
 "SourceGroup": "172.31.128.0/24"
},
{
 "SecurityGroup": "DB",
```

```
      "RuleType": "inbound",
      "Protocol": "tcp",
      "Port": "1434",
      "SourceGroup": "172.31.16.0/24"
     },
     {
      "SecurityGroup": "DB",
      "RuleType": "inbound",
      "Protocol": "tcp",
      "Port": "3389",
      "SourceGroup": "172.31.0.0/16"
     },
     {
      "SecurityGroup": "ELB",
      "RuleType": "inbound",
      "Protocol": "all",
      "Port": "all",
      "SourceGroup": "172.31.0.0/16"
     }
   ]
},
{
  "VPNFlag": "Disable"
},
{
  "DNSEntriesFile": "assignment1/awsvpcb.assignment1.DNS.json"
},
{
  "ELBs": [
    {
      "ELBName": "rainforest",
      "ListenerProtocol": "HTTPS",
      "ListenerPort": "443",
      "InstanceProtocol": "HTTP",
      "InstancePort": "80",
      "ELBSubnet": "ELB",
      "HealthCheckTarget": "TCP:80",
      "HealthCheckInterval": "5",
      "HealthCheckTimeout": "3",
      "HealthCheckUnhealthyThreshold": "2",
      "HealthCheckHealthyThreshold": "2",
      "EnableSessionStickiness": "Y",
      "ELBInstances": [
        {
          "InstanceName": "LINUX1"
        },
```

```
                {
                 "InstanceName": "LINUX2"
                }
               ]
              }
             ]
            }
}
```

ASSIGNMENT DNS JSON FILES

The awsvpcb-scripts.zip file includes sample assignment DNS JSON configuration files in the "secfiles/assignment#" directories (#=1-3). AWSVPCB allows for each assignment to include its own set of DNS entries for the servers and or other components. The DNS JSON file must be configured within the assignment JSON file. Thus, there could be one centralized DNS JSON config file used for multiple assignments, if desired. The format of the DNS JSON configuration file is dictated by AWS as it is loaded directly without modification. Any parameters accepted by AWS would be accepted in this file. Please refer to https://docs.aws.amazon.com/cli/latest/reference/route53/change-resource-record-sets.html for available JSON elements.

ONE OF THE THREE SAMPLE AWSVPCB ASSIGNMENT DNS JSON INCLUDED IN ZIP FILE (secfiles/assignment1/awsvpcb.assignement1.DNS.json)

```
          {
           "Changes": [
            {
             "Action": "CREATE",
             "ResourceRecordSet": {
              "Name": "nat.awsvpcb.edu",
              "Type": "A",
              "TTL": 300,
              "ResourceRecords": [
               {
                "Value": "172.31.132.151"
               }
              ]
             }
            },
            {
             "Action": "CREATE",
             "ResourceRecordSet": {
              "Name": "iis1.awsvpcb.edu",
              "Type": "A",
              "TTL": 300,
              "ResourceRecords": [
               {
                "Value": "172.31.128.98"
```

```
      }
     ]
    }
   },
   {
    "Action": "CREATE",
    "ResourceRecordSet": {
     "Name": "iis2.awsvpcb.edu",
     "Type": "A",
     "TTL": 300,
     "ResourceRecords": [
      {
       "Value": "172.31.128.99"
      }
     ]
    }
   },
   {
    "Action": "CREATE",
    "ResourceRecordSet": {
     "Name": "linux1.awsvpcb.edu",
     "Type": "A",
     "TTL": 300,
     "ResourceRecords": [
      {
       "Value": "172.31.128.43"
      }
     ]
    }
   },
   {
    "Action": "CREATE",
    "ResourceRecordSet": {
     "Name": "linux2.awsvpcb.edu",
     "Type": "A",
     "TTL": 300,
     "ResourceRecords": [
      {
       "Value": "172.31.128.44"
      }
     ]
    }
   },
   {
    "Action": "CREATE",
    "ResourceRecordSet": {
```

```
            "Name": "mssql.awsvpcb.edu",
            "Type": "A",
            "TTL": 300,
            "ResourceRecords": [
              {
                "Value": "172.31.129.75"
              }
            ]
          }
        }
      ]
}
```

13.5 Havoc Circus Documentation

Havoc Circus is a set of C#.Net packages used to run the HavocCircus Windows service which allows you to dynamically change your VPC based on configuration files in JSON format. Havoc Circus was designed to be highly flexible such that almost any scenario can be created within your VPC. Havoc Circus must be installed on a Windows Server-based AMI that is part of an assignment as built by AWSVPCB within your VPC. Havoc Circus is pre-installed within the mssql AMI (one of the three publicly available Clouducate AMIs). Havoc Circus also requires a SQL Server database called HavocMonkey to be available within your VPC – the mssql sample AMI also includes this database.

Havoc Circus Components

Havoc Circus includes the following components: CTS4743 Havoc Circus Windows Service with associated JSON configuration file & the HavocMonkey SQL Server database, HavocCircus assignment directories & associated configuration JSON configuration files, and the sample AMIs that provide two sample web applications as well as two sample batch jobs. Also, these components integrate with the AWSVPCB scripts in several ways. This integration is called out as each component is reviewed below.

CTS4743 Havoc Circus Windows Service

The CTS4743 Havoc Circus Windows service is installed by default in the *C:\Program Files\CTS 4373\Services\Havoc Circus* directory. If you create your own image, it can be installed in any directory, however, you need to make sure that all the JSON configuration files are also adjusted accordingly. This directory contains all the files needed to run the Windows service, although the HavocMonkey database also needs to be available somewhere within your VPC. Of particular note is the *HavocCircus.exe.config* file which is a JSON formatted configuration file for the service. Note that there is also a *staging* directory used during the assignment modifications and this is necessary as well.

HavocCircus.exe.config

There is only one configuration file for the Havoc Circus Windows service. This is a standard .NET configuration file. Thus, the modifiable values are in the "appSettings" section. All of the "keys" in this section are required. Below is a description of what these keys are used for:

- connectionString: This is the SQL Server connection string to the HavocMonkey database. The default refers to the database that comes pre-installed with the mssql AMI.
- manifestFileName: The default is manifest.json. This is the suffix for the assignment manifest file that HavocCircus will load after appending the assignment name at the front (e.g., assignment1.manifest.json).
- processExecutionThreadSleepTime: This is the number of seconds that the service will sleep in between cycles as it wakes to see if the tasks have completed.
- s3BucketName: This is the name of the S3Bucket where the manifest and all associated assignment support files are located.
- s3IAMAccessKey: This is the AWS IAM Access Key used to retrieve the manifest and all associated assignment support files from the designated S3 Bucket.
- s3IAMSecretKey: This is the AWS IAM Secret Key used to retrieve the manifest and all associated assignment support files from the designated S3 Bucket.
- scpClientExecutable: This is the location of the executable used to copy files to any Linux system within your VPC.
- sshClientExecutable: This is the location of the executable used to execute commands on any Linux system within your VPC.
- sshPrivateKey: This is the private key used to authenticate to any Linux system within your VPC.
- startupProcessingDelayThreadSleepTime: This is how many milliseconds HavocCircus will wait after starting up before executing the HavocCircus commands from the manifest file.
- triggerTXTRecordName: This is the DNS name of the TXT record populated by AWSVPCB to let HavocCircus know the assignment that is executing. This is not easily changeable at this time as it is not variable within the AWSVPCB scripts yet. Thus, it must remain as the default trigger.awsvpcb.edu.
- workingDirectory: This is the directory where all the HavocCircus files and staging directory is located.

SAMPLE HAVOCCIRCUS.EXE.CONFIG JSON INCLUDED ON THE <u>MSSQL</u> AMI (C:\Program Files\CTS 4373\Services\Havoc Circus\HavocCircus.exe.config)

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
  <appSettings>
    <add key="connectionString" value="Data Source=mssql.awsvpcb.edu;Integrated
Security=False;Initial Catalog=HavocMonkey;User
ID=SvcAcc_CTS4743_HavocCircus;Password=***" />
```

```
    <add key="manifestFileName" value="manifest.json" />
    <add key="processExecutionThreadSleepTime" value="50" />
    <add key="s3BucketName" value="public.awsvpcb.edu" />
    <add key="s3IAMAccessKey" value="***" />
    <add key="s3IAMSecretKey" value="***" />
    <add key="scpClientExecutable" value="C:\Program Files (x86)\PuTTY\pscp.exe"
/>
    <add key="sshClientExecutable" value="C:\Program Files (x86)\PuTTY\plink.exe"
/>
    <add key="sshPrivateKey" value="C:\Program Files\CTS 4373\Services\Havoc
Circus\privkey.ppk" />
    <add key="startupProcessingDelayThreadSleepTime" value="10000" />
    <add key="triggerTXTRecordName" value="trigger.awsvpcb.edu." />
    <add key="workingDirectory" value="C:\Program Files\CTS 4373\Services\Havoc
Circus\" />
  </appSettings>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Buffers"
publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
        <bindingRedirect oldVersion="0.0.0.0-4.0.3.0" newVersion="4.0.3.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

HavocMonkey Database

The HavocMonkey database only contains 2 tables and is only used to track whether a HavocCircus task was completed or not. The two tables are “AssignmentTasks” and “States”. These should not be to be changed.

Havoc Circus Assignment JSON Files

The core of the Havoc Circus functionality is controlled through the Havoc Circus Assignment JSON files. These files tell Havoc Circus what to do for any particular assignment and allow for a tremendous amount of flexibility. We provide some examples of this with assignments 1-3 that do not cause any application harm but provide a general idea of what is possible.

First, let’s take a look at the sections available in a Havoc Circus JSON file. There are three sections available to be used in Havoc Circus JSON files. The first section is “PrimateAssembly” and it does not need to be touched as this is legacy code that will be retired in the future. The other two sections, which are adjustable, are “PrimatePackages” and “PrimateTasks”. The PrimatePackages section is optional and includes all the files that reside in the S3 bucket directory for the assignment that needs to be downloaded to a specific server within your VPC such that they can be later used by a Primate Task. You would generally have one PrimatePackage section per destination server. PrimateTasks are the commands you would

like Havoc Circus to execute against specific servers within your VPC. As with PrimatePackages, a PrimateTask is typically aimed at one server, so you would at least need one per destination server. You can have multiple PrimateTasks for a particular destination server if desired for some reason (e.g., you need one task to complete before starting the next). Now, let's review the JSON parameters for each section.

SECTION – PrimatePackages (optional and there can be multiple)
TargetHostName: The DNS name or IP of the host you wish to receive the file(s).
TargetUserName: The username to use to log into the destination host.
TargetUserPassword: The password for the username to use to log into the destination host.
TargetPlatform: This can be either "Windows" or "Linux".
TargetPackageFiles: This is a collection of files (one or many) to copy to the destination server.
TargetPackageFiles-FileName: The name of the file to pick up from the S3 Bucket assignment directory and copy to the destination host.
TargetPackageFiles-FilePath: The file path on the destination where the file is to be copied.

SECTION – PrimateTasks (required and there can be multiple)
AlwaysExecuteOnStartup: This would be "true" or "false". If true, then every time that the Havoc Circus Windows Service starts, it will execute this Primate Task.
TaskLabel: This is simply a label for tracking whether the task was executed already within the HavocMonkey database.
TargetHostName: The DNS name or IP of the host onto which you wish to execute the command(s).
TargetUserName: The username to use to log into the destination host where the command(s) will be executed.
TargetUserPassword: The password for the username to use to log into the destination host where the command(s) will be executed.
TargetPlatform: This can be either "Windows" or "Linux".
TaskInitializationCommands: This is a collection of commands (one or many) to execute on the destination server.
TaskInitializationCommands-CommandExecutable: The name of the executable to run on the destination server. On Linux, this parameter will also include the arguments to pass into the command, but on Windows the arguments need to be passed through a separate JSON parameter (CommandArguments) mentioned below.
TaskInitializationCommands-CommandArguments: Typically, only needed on Windows. These would be the arguments to pass into the CommandExcecutable run on the destination server.

SAMPLE HAVOC CIRCUS ASSIGNMENT JSON FILE


```
{
"HavocMonkey": {
  "PrimateAssembly": {
   "PrimateTypeName": "Assignment3.HavocMonkey",
   "PrimateFileName": "Assignment3.HavocMonkey.dll",
   "PrimateDependencies": [
    {
     "DependencyFileName": "Primate.dll"
    }
   ]
  },
  "PrimatePackages": [
   {
    "PrimatePackage": {
     "TargetHostName": "iis1.awsvpcb.edu",
     "TargetUserName": "Administrator",
     "TargetUserPassword": "****",
     "TargetPlatform": "Windows",
     "TargetPackageFiles": [
      {
       "FileName": "Looper.exe",
       "FilePath": "C$\\Apps\\Looper\\"
      },
      {
       "FileName": "Looper.dll",
       "FilePath": "C$\\Apps\\Looper\\"
      }
     ]
    }
   },
   {
    "PrimatePackage": {
     "TargetHostName": "linux1.awsvpcb.edu",
     "TargetUserName": "ec2-user",
     "TargetUserPassword": "****",
     "TargetPlatform": "Linux",
     "TargetPackageFiles": [
      {
       "FileName": "Looper.exe",
       "FilePath": "/usr/bin/cts4743/apps/looper/"
      },
      {
       "FileName": "Looper.dll",
       "FilePath": "/usr/bin/cts4743/apps/looper/"
      }
```


LEAVE THIS SECTION "AS IS"

```
          ]
         }
        }
       ],
       "PrimateTasks": [
        {
         "PrimateTask": {
          "AlwaysExecuteOnStartup": true,
          "TaskLabel": "Looper1",
          "TargetHostName": "iis1.awsvpcb.edu",
          "TargetUserName": "Administrator",
          "TargetUserPassword": "*****",
          "TargetPlatform": "Windows",
          "TaskInitializationCommands": [
           {
            "CommandExecutable": "C:\\Apps\\Looper\\Looper.exe",
            "CommandArguments": "33"
           }
          ]
         }
        },
        {
         "PrimateTask": {
          "AlwaysExecuteOnStartup": true,
          "TaskLabel": "Looper2",
          "TargetHostName": "linux1.awsvpcb.edu",
          "TargetUserName": "ec2-user",
          "TargetUserPassword": "****",
          "TargetPlatform": "Linux",
          "TaskInitializationCommands": [
           {
            "CommandExecutable": "\"`nohup dotnet
  \"/usr/bin/cts4743/apps/looper/Looper.dll 125\" > /dev/null 2>&1 &`\"",
            "CommandArguments": ""
           },
           {
            "CommandExecutable": "\"`nohup dotnet
  \"/usr/bin/cts4743/apps/looper/Looper.dll 125\" > /dev/null 2>&1 &`\"",
            "CommandArguments": ""
           }
          ]
         }
        }
       ]
      }
}
```

}

Havoc Circus Sample AMIs
Havoc Circus and AWSVPCB come with 4 publicly available AMIs. These AMIs will change over time, so they are updated in our publicly available AWS S3 Bucket ***public.awsvpcb.edu***. The AWSVPCB VPC and ASSIGNMENT JSON files will be updated within this bucket as new AMIs are published and the code is changed. Note that the administrator passwords for all AMIs are included in the secfiles directory of the AWSVPCB scripts as well as in the sample Havoc Circus assignment JSON files. Below is a summary of the 4 publicly available AMIs and what each one contains.

NAT Instance AMI – used by nat.awsvpcb.edu
This is the simplest of the AMIs as it contains only configuration information to allow for Internet access out of your VPC. This AMI participates in the public routing table, while all other AMIs participate in the private routing table. This AMI contains both a private IP and a public IP. It is the only AMI with a public IP and it routes Internet traffic out the VPC's Internet Gateway.

LINUX AMI – used by linux1.awsvpcb.edu and linux2.awsvpcb.edu
This AMI includes two key services – nginx & kestrel-rainforest. Both of these services can be started/stopped/restarted using systemctl as root (e.g., sudo systemctl start kestrel-rainforest). The nginx service listens on port 80 and its configuration is set up to proxy all traffic to the kestrel-rainforest service that listens on port 5000. The kestrel-rainforest service hosts a dotnet application called rainforest that is an implementation of Microsoft's eShopOnWeb (https://github.com/dotnet-architecture/eShopOnWeb). The kestrel-rainforest service is configured to utilize a couple of databases on the mssql.awsvpcb.edu server. This application requires session stickiness on the load balancer to perform properly as exampled on assignment1.

This AMI also includes a batch job that can be used for assignments. The batch job is located at */var/batch/catalog_exporter* on the AMI. This batch job creates an export of the items available on the site and can be executed with the following command: *dotnet CatalogExporter.dll* . The output of this job is sent to the export subdirectory. This batch job is also configured to utilize the same database on the mssql.awsvpcb.edu server.

IIS AMI – used by iis1.awsvpcb.edu and iis2.awsvpcb.edu
This AMI includes IIS and an implementation of Microsoft's Contoso University web application (https://github.com/dotnet/AspNetCore.Docs/tree/master/aspnetcore/data/ef-mvc/intro/samples). IIS is configured with just one site and only has one app pool running. The Contoso University web application is configured to utilize a database on the mssql.awsvpcb.edu server. IIS listens on both ports 80 and 5000. This application does not require session stickiness on the load balancer to perform properly as exampled on assignment2.

This AMI also includes a batch job that can be used for assignments. The batch job is located at *C:\Batch\StudentGenerator* on the AMI. As the name implies, the batch job randomly

generates students and imports them into the Contoso University database. The batch job accepts a parameter to indicate the number of students to generate. This batch job is also configured to utilize the same database on the mssql.awsvpcb.edu server.

MSSQL AMI – used by mssql.awsvpcb.edu
This AMI includes the CTS4743 Havoc Circus Windows service and SQL Server 2019 to contain 4 databases (2 for eShopOnWeb, 1 for Contoso University, and 1 for Havoc Circus). SQL Server listens on the default 1433 port, however, the DAC Admin port TCP 1434 is also enabled and available within the VPC depending on the firewall rules created by the AWSVPCB assignment JSON. The sa password for this instance is included in the secfiles directory of the AWSVPCB scripts.

13.6 CLOUDUCATE GETTING STARTED

You can use the default awsvpcb-scripts provided to build a VPC with VPN connectivity that includes publicly available Havoc Circus AMIs with one or two applications loaded. This will use static JSON files pre-loaded in the awsvpcb "secfiles" directory for the VPC and Assignments (1-3 are included) to provide you with examples of what can be configured with AWSVPCB. This will also use static assignment JSON files loaded on the Havoc Circus AMIs that provide examples of what you can do with Havoc Circus.

GETTING STARTED PROOF OF CONCEPT (POC) PROCEDURE

1. CREATE TEST STUDENT AWS ACCOUNT - This will be where the VPC is created
    - NOTE: At this time, this cannot be an AWS Educate Starter account as such accounts do not support VPN connectivity
    - NOTE: Credit card must be provided. AWS credits should be obtained and added to this account to avoid unnecessary charges. However, the charges for going through this POC are minimal (under $10) as long as everything is destroyed in the end and it is not kept up and running for hours.
2. DECIDE which PC you are planning to use to access your VPC (Linux, MAC, or Windows) through the VPN.
3. INSTALL OpenVPN on selected PC - This will be used later when the OVPN file is created.
    - ON MAC: OpenVPN Connect version 3.1 or higher from the MAC App Store
    - ON WINDOWS: https://www.ovpn.com/en/guides/windows-openvpn-gui
4. DECIDE where you are going to run the scripts (must be a Linux or MAC machine) - if using MAC or Linux for the PC where OpenVPN is installed, then it can be the same machine.
5. INSTALL the AWS CLI (version 2) on the chosen machine where the scripts will run - follow instructions provided by AWS. (https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html) and test with the following command to make sure you have the proper access: aws ec2 describe-instances
6. DOWNLOAD the awsvpcb-scripts file at https://github.com/clouducate/awsvpcb/blob/main/awsvpcb-scripts.zip and EXTRACT onto the machine where the AWS CLI is installed.
7. CD to the root of the awsvpcb-scripts directory.

8. RUN ./AWSVPCB.CONFIGURE
9. RUN ./AWSVPCB.TEST to confirm all is working as expected
10. RUN ./AWSVPCB.VPC.CREATE
11. IF NECESSARY, DOWNLOAD the AWSVPCB-client-config.ovpn file created in the "secfiles" directory by the AWSVPCB.VPC.CREATE script to the PC where you installed the OpenVPN client. If the scripts reside on the same machine as your OpenVPN client, then this is unnecessary.
12. IMPORT the AWSVPCB-client-config.ovpn into the OpenVPN client.
13. IF NECESSARY, DOWNLOAD the ca.crt file in the "secfiles" directory to the PC you installed the OpenVPN client. Again, if the scripts reside on the same machine as your OpenVPN client, then this is unnecessary.
14. ADD the ca.crt root certificate to the trusted certificate store on your PC (Windows, MAC, or Linux).
15. IF NECESSARY, DOWNLOAD other support files from the "secfiles" directory to the PC including:
    - iis1.rdp, iis1.password, iis2.rdp, iis2.password, mssql.rdp, mssql.password, mssql.sa.password, privkey.ppk (if you plan to use putty), privkey.pem (if you plan to use SSH)
13. RUN ./AWSVPCB.ASSIGNMENT.CREATE 1
14. RUN ./AWSVPCB.ASSIGNMENT.START
15. CONNECT with OpenVPN to your AWS VPC using the OVPN connection you just imported.
16. TEST the following:
    - RDP to the iis1.awsvpcb.edu server using the iis1.rdp file in the "secfiles" directory. The password is in the iis1.password file. If using a MAC for the VPN connectivity, then download Microsoft Remote Desktop 10 or higher from the MAC App Store.
    - RDP to the mssql.awsvpcb.edu server using the mssql.rdp file in the "secfiles" directory. The password is in the mssql.password file. If using a MAC for the VPN connectivity, then download Microsoft Remote Desktop 10 or higher from the MAC App Store.
    - SSH to the linux1.awsvpcb.edu server using the privkey.pem file (password is hardcoded to cts4743) or use putty (need to add privkey.ppk to the session definition - same password of cts4743).
    - Use SSMS (Windows) or Azure Data Studio (MAC) to connect to the SQL Server instance on mssql.awsvpcb.edu. You can log in as “sa” using the password in the mssql.sa.password file.
    - Use a browser to connect to https://rainforest.awsvpcb.edu (assignment 1) or https://myfiu.awsvpcb.edu (assignment 2) or http://myfiu.awsvpcb.edu AND http://rainforest.awsvpcb.edu (assignment 3). **NOTE**: Assignments 1 & 2 use the AWS ELB and thus utilize TLS (https) and http is disabled.
17. RUN ./AWSVPCB.ASSIGNMENT.STOP # this would only be necessary if you wanted to get back to this later, else you can run ./AWSVPCB.ASSIGNMENT.DESTROY

NEXT STEPS FOR YOUR COURSE

1. MAP OUT TARGET ENVIRONMENT
    - What AMIs do you plan to use? Are the default AMIs provided sufficient or do you want to take those and create custom AMIs?
    - What applications will the students be testing? Are the default rainforest and myfiu applications sufficient or do I want to build/use others?
    - What do you want your VPC to look like (IP range, subnets, possible instances, possible ELBs)?
    - What do you want your assignments to look like (instances, firewall rules, DNS entries, ELBs)? NOTE: To use Havoc Circus, at least one of the AMIs must be Windows with the Havoc Circus service pre-loaded and pointing to your Havoc Circus manifest location locally on the server or in AWS.
    - What do you want your assignments to do (e.g., set up a security vulnerability or break the environment in some way)?
2. CHOOSE DOMAIN
    - Use default awsvpcb.edu domain - This limits you to two ELB names (myfiu and rainforest), but other than that, there are no other limitations OR
    - Create your own domain and provide the following:
        - Public CA cert for import into student client machines or use public CA (e.g., Sectigo)
        - Client & Server VPN certs and private keys
        - ELB certs and private keys
        - All of these would need to be placed in the "secfiles" directory with the appropriate name (NOTE: the VPN server and client names are currently hardcoded, so you need to replace the files in the secfiles directory)
3. SETUP INSTRUCTOR AWS ACCOUNT (optional). This would be needed if you wanted to do any of the following:
    - Host custom AMIs for the assignments that will not be publicly available
    - House Havoc Circus assignment dependency files and manifest JSON files
    - House AWSVPCB VPC and assignment JSON files
    - If you wish to enable AWS logging, then you would need an IAM user with Cloudwatch access
    - If you wish to be able to gather diagnostic information, then the same IAM user needs access to be able to create a new S3 bucket
4. CREATE your VPC and assignment AWSVPCB JSON files.
5. CREATE Havoc Circus assignment JSON files.
6. MODIFY your vpcb-config as appropriate.
7. TEST a lot.
8. CREATE instructions for students to create their own AWS account.
9. PROVIDE AWS credits and preferably a shared/University controlled Linux server from which the students can run the AWS CLI client.
10. IF USING CUSTOM/PRIVATE AMIs, the grant access to students' AWS accounts.