AWS Control Tower Adoption Strategies

This document describes some of the methods the AWS development team at the University of Virginia has used to build out a multi-account architecture using AWS Organizations with Control Tower.

Problem:

Create a centrally controlled AWS account infrastructure that allows for provisioning of new accounts for client departments and users, with various compliance regimens optionally applied. Client accounts should be housed under a central Organization, and the Organization administrators should have centralized visibility into client accounts for the purposes of providing new services, monitoring, security, and billing. Compliant accounts should share certain data with the central management account set, such as server logs, API calls, network traffic data, and so forth for the purposes of compliance monitoring and possible audit.

AWS Control Tower is the primary solution for the multi-account offering, but in its current incarnation, it has a number of limitations that require workarounds or enhancements. Some example limitations are:

- Inability to define new client account VPC CIDRs and Subnets on a per-account basis. All new accounts – if provisioned with a default VPC – get the same CIDR. This will cause issues if those VPCs ever have to talk to each other or back campus networks
- Limited set of GuardRails and no ability to edit/modify GuardRails or add new GuardRails. (AWS is working on custom GuardRail support)
- No built-in system for provisioning services to new accounts via Control Tower and Service Catalog, or sharing centralized AMIs, encryption keys, and so forth.

Strategies:

Account type separation by OU: Control Tower uses OU-based account separation, such that different GuardRails may be assigned to different OUs. Different compliance regimens, such as HIPAA or NIST800-171, will be assigned their own OUs, with specific GuardRails, Config Rules, and so forth applied as necessary. In addition, CloudFormation mechanisms allow for deployment of StackSet instances only to specific OUs, enabling mass deployment of regimen-specific Stacks to only the OUs that need them.

Customizations for Control Tower Pipeline: The recently released “Customizations for Control Tower” solution from AWS (https://aws.amazon.com /solutions/customizations-for-aws-control-tower) creates a number of resources in the Control Tower master account to support an event-driven CodePipeline application. New accounts provisioned with the Account Factory trigger the pipeline to deploy SCPs and CloudFormation StackSets to specific accounts or OUs. As of this writing, we are just starting to work this into our provisioning chain, but it has proven to manage a number of provisioning tasks that we were previously doing manually or attempting to build automation for.

Life cycle event capturing for automation: AWS Organizations generates various CloudTrail Events when child accounts are created or moved into new OUs. For provisioning needs that cannot be handled by the Customizations pipeline, we capture these events using AWS Event Bridge (which encompasses CloudWatch Event Rules), and use them to drive automated provisioning -– via tools such as SNS and Lambda – of resources that are specific to the account or OU referenced in the Event, such as KMS key shares, AMI shares, Transit Gateway connections, Config Rules, and GuardDuty membership. Over time, we may determine how to fit many of these into the pipeline.

Transit Gateway for simplified VPN control: The AWS Transit Gateway gives us simplified control over child account VPN connections back to our campus networks, and obviates the need for us to coordinate with our Networks team to provide VPN connections for new child accounts. The Transit Gateway has a pair of redundant tunnels to each of our two data centers. Adding a child account VPC as a Transit Gateway consumer takes several minutes to configure, and involves adding configuration on both the Transit Gateway end, and the child account end. Much of this can be automated via CloudTrail Event monitoring. One caveat for VPCs that connect to the Transit Gateway: their network CIDRs must be within a range allotted by our Networks team that does not overlap with the CIDRs of any on-campus networks or other VPCs that route to the Transit Gateway.

StackSets for CloudFormation deployments at the OU level: CloudFormation StackSets allow for deployment to entire OUs. This forms the basis of our automation strategy whereby OU-specific Stacks (such as those containing Config Rules or other security/compliance specific resources) can be deployed via the Customizations pipeline, or independent Event triggers to specific accounts or OUs.

Aviatrix Gateways for compliant Transit Gateway connections: Compliant accounts may have more substantial security requirements for external access by their private subnets, and so we use Aviatrix Gateways in place of the standard AWS NAT gateways. This allows us to allow or deny certain destinations, and have more granular control over traffic coming from compliant subnets. Since there is an increased cost associated with using the Aviatrix Gateways versus standard NAT instances, they are only deployed as necessary per a given account.

SCPs (Service Control Policies) to limit user access and actions to specific regions: Unlike Config Rules that can only detect configuration or compliance violations, Service Control Policies can completely restrict access to certain regions, services, or actions at the Organization or OU level. Control Tower already uses SCPs for their ‘preventative’ GuardRails, but we layer over additional SCPs to both prevent user access to certain regions and control costs. New SCPs can be easily deployed and managed through the Customizations pipeline solution.

AWS advises that Config Recorders be enabled in all active regions, but this incurs a cost, especially if most regions go unused. GuardDuty, used for API security monitoring, is also advised to be enabled in all regions. To minimize the associated costs, we use an SCP to effectively disable access to all AWS services outside of the regions in use by Control Tower. Then we deploy GuardDuty and Config Rules to *only* the regions within Control Tower’s scope, thus lowering the cost of those services. In addition, we deploy an SCP to compliant OUs that limits our SSO users to access only us-east-1. These SCPs can be modified at any point from the Master Payer account, and access changes take effect immediately.

GuardDuty for API security monitoring: GuardDuty monitors API calls in a given region, classifying the potential severity of suspicious API calls and giving insights through a dashboard. New child accounts are automatically enrolled in GuardDuty in the Control Tower regions and connected to the GuardDuty “master” in the Audit account. This is done through event-based triggering of a Lambda function.

AMI build pipeline to share images to client accounts: Our current build pipeline uses GitLab CI/CD to build images using Packer and Ansible on GitLab Runners running on EC2 instances in our Dev and Prod environments. A Git push to the staging branch of our build pipeline will deploy a new image to the Dev environment, while a merge/push to master branch will build and deploy to our Prod environment. When an AMI is successfully built, the build script looks up all of the active accounts in the current Org, and shares the AMI out to those accounts. It then updates an SSM Parameter, “GoldImage”, in all of the accounts to point to the new AMI. In this way, the Service Catalog EC2 product can just point to the “GoldImage” parameter, and always be assured of running the latest AMI. **Note***: We are currently working to transition the AMI pipeline to using AWS CodeCommit, CodeBuild, and CodePipeline instead of GitLab. The general build/deploy/share process though is the same.*

AWS Service Catalog for provisioning new services and instances: The Service Catalog allows us to more granularly control what services clients can use, and in some cases, constrain what settings those services can be launched with. For example, we can limit users, through RBAC, to only launching instances with the Service Catalog, and furthermore, we can limit what types of instances, AMIs, etc. can be launched. As we build out our platform, new client products will be offered through the Service Catalog.

CloudWatch Agent on instances for cross-account logging and alarming: Instances from our shared AMI are built to self-configure the CloudWatch agent when they launch, which enables them to stream log files and metrics to CloudWatch Logs in the master account. This is accomplished through EC2 instance profiles in the child accounts that have assume-role privileges into a CloudWatch role in the Master account. This is particularly important for compliant accounts, where we need to ensure that users of those accounts cannot modify/delete log files. One drawback at this point is that logs sent to the master account are not sent to CloudWatch in the child account, and so we cannot currently create Alarms on those log files that would trigger in the child account.