GRACE Cloud Platform

$\underbrace{\text{GRACE}}$ ↓

GSA Readily Available Computing Environment

DevSecOps Working Group

GSA IT

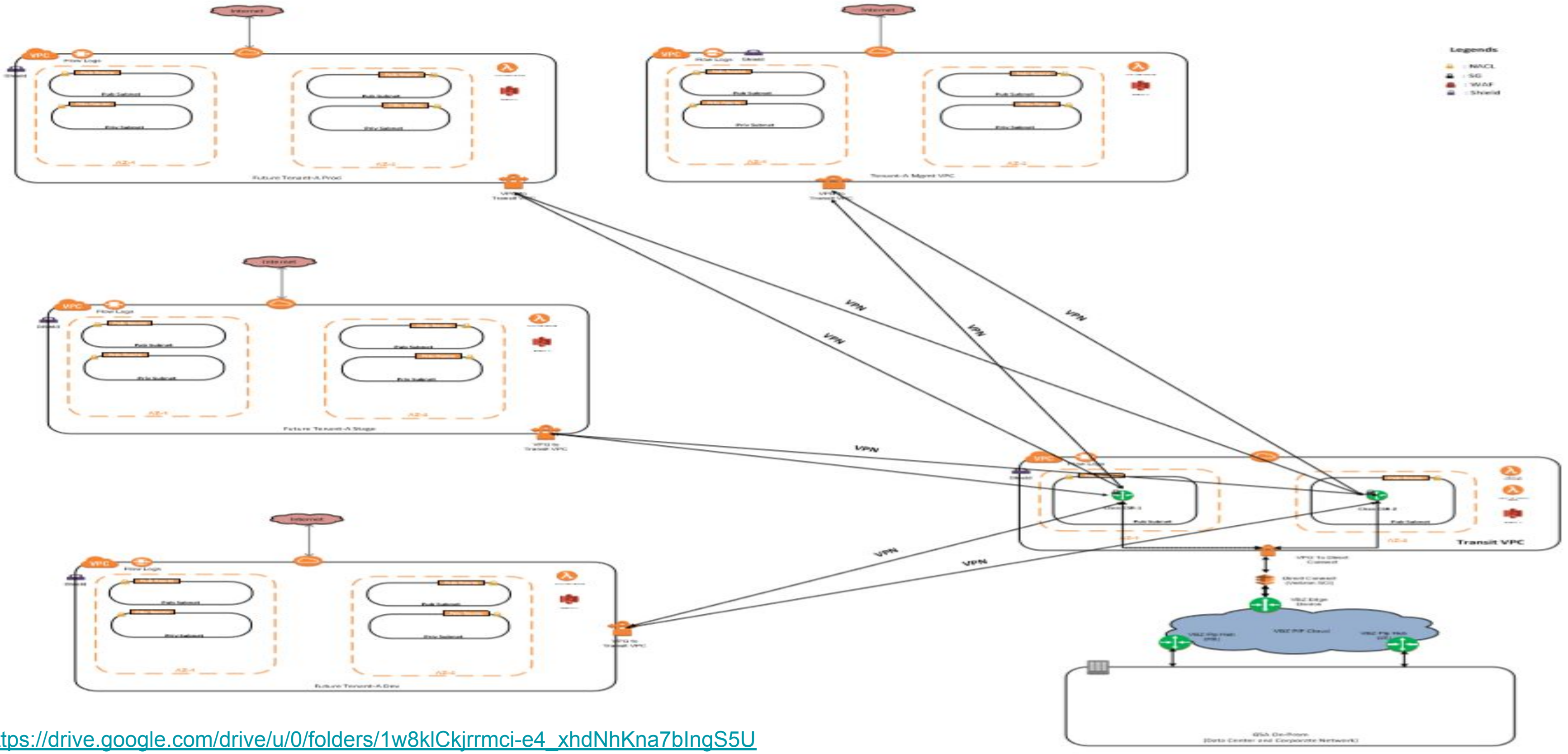
AGENDA

- GRACE Core Principles
- GRACE Network Architecture
- GRACE Account Provisioning Workflow
- GRACE User Provisioning Workflow
- GRACE Incremental LATO Process
- GRACE Components
- GRACE Leveraged Services
- GRACE GitHub Repositories
- GRACE Points of Contact
- Questions ?

GRACE Core Principles

- **Automation**
  - IaC - Infrastructure as Code or programmable infrastructure
  - Programmable infrastructure, means writing code (which can be done using a high level language or any descriptive language) to manage configurations and automate provisioning of infrastructure in addition to deployments
  - Scripting environments — from installing an operating system, to installing and configuring servers, to configuring how the instances and software communicate with one another
  - I*nfrastructure automation*, which involves replicating steps multiple times and reproducing them on several servers
- **Self-Service Provisioning**
  - Customers (tenants) manage the level of control and speed at which they can spin up machines
  - Tenant owners control/manage their infrastructure costs
  - Standard automation code (scripts) to launch/provision services will be made available to tenants to launch infrastructure components in their tenant accounts
  - Tenants are not constrained and hardware resources are readily available for automated provisioning

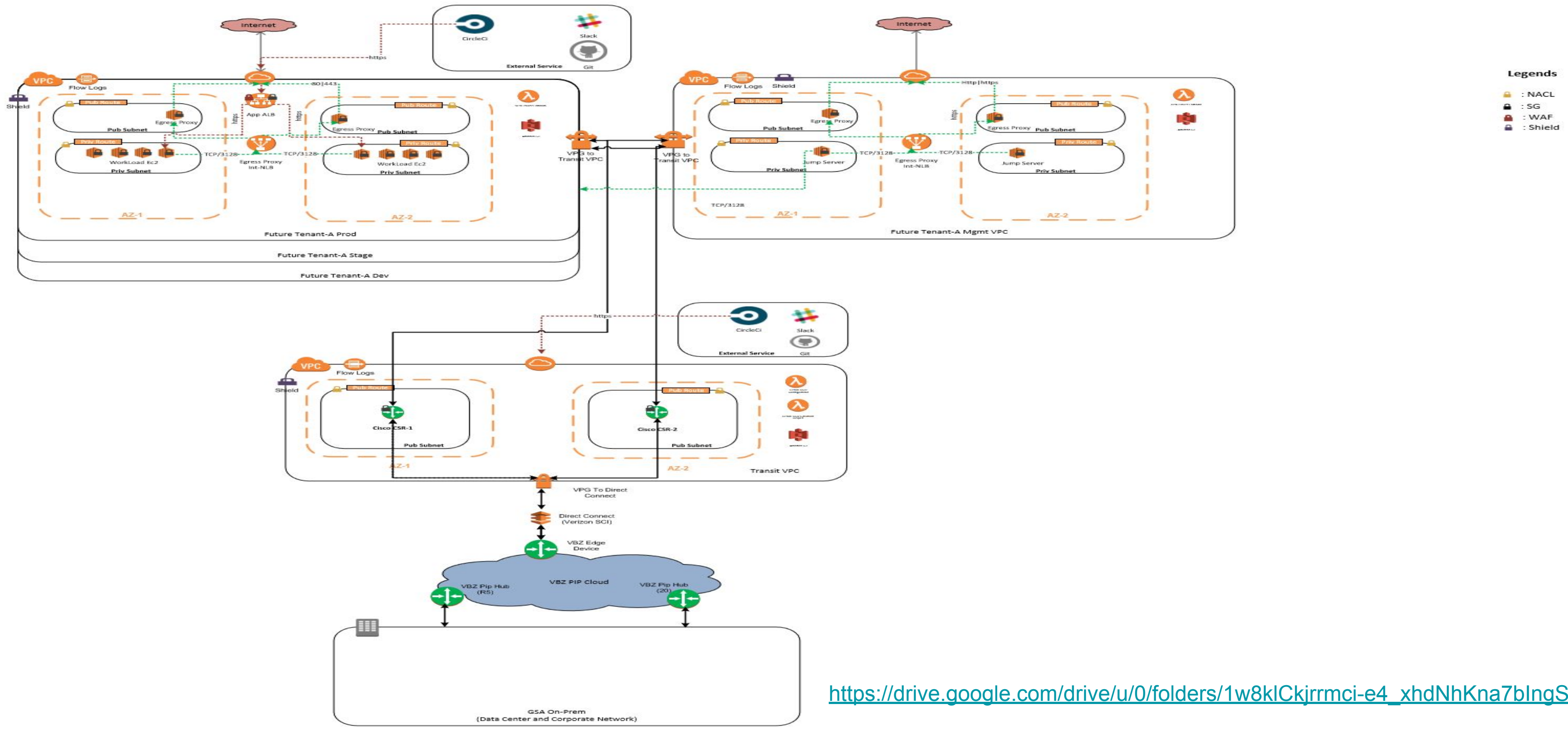
GRACE Network Architecture



https://drive.google.com/drive/u/0/folders/1w8klCkjrrmci-e4_xhdNhKna7bIngS5U

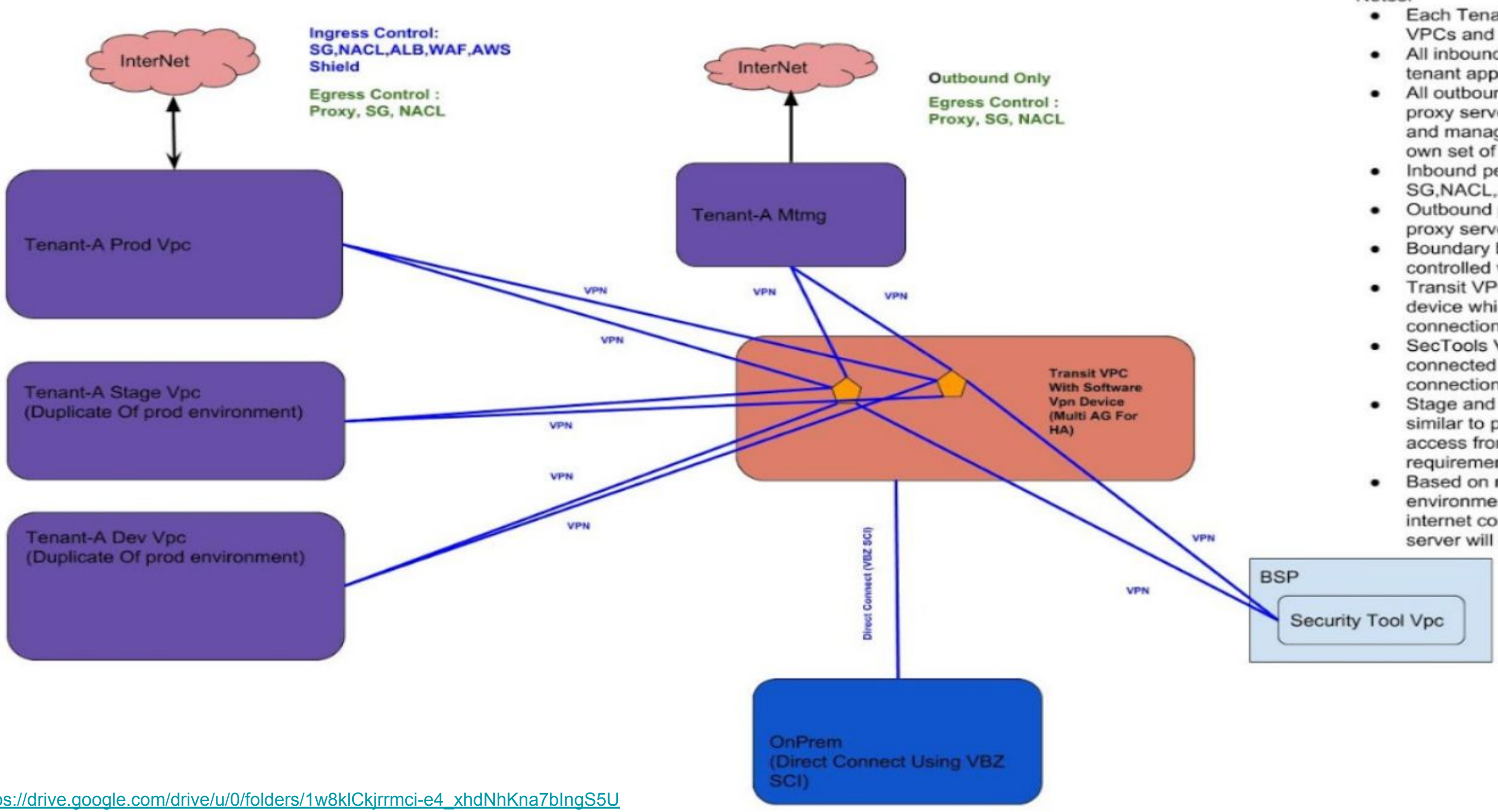
GSA IT

GRACE Network Architecture



https://drive.google.com/drive/u/0/folders/1w8klCkjrrmci-e4_xhdNhKna7bIngS5U

GRACE Network Architecture



Notes:

- Each Tenant will have their application VPCs and management vpc
- All inbound application traffic will go to tenant application vpc
- All outbound web traffic will go through proxy server. Tenant application vpc and management vpc will have their own set of proxy servers.
- Inbound perimeter control will use SG,NACL,ALB,WAF and AWS Shield
- Outbound perimeter control will use proxy server with url filtering and SG
- Boundary between tenant vpc will be controlled with Nacl and SG
- Transit VPC will have software VPN device which will connect all vpc and connection back to on-prem.
- SecTools VPC at BSP will be connected through transit vpc with vpn connection to bsp
- Stage and Dev environment will be similar to prod environment. Application access from internet will be based on requirement for lower environment.
- Based on requirement , lower environment may only have outbound internet connection through proxy server will ssl decryption

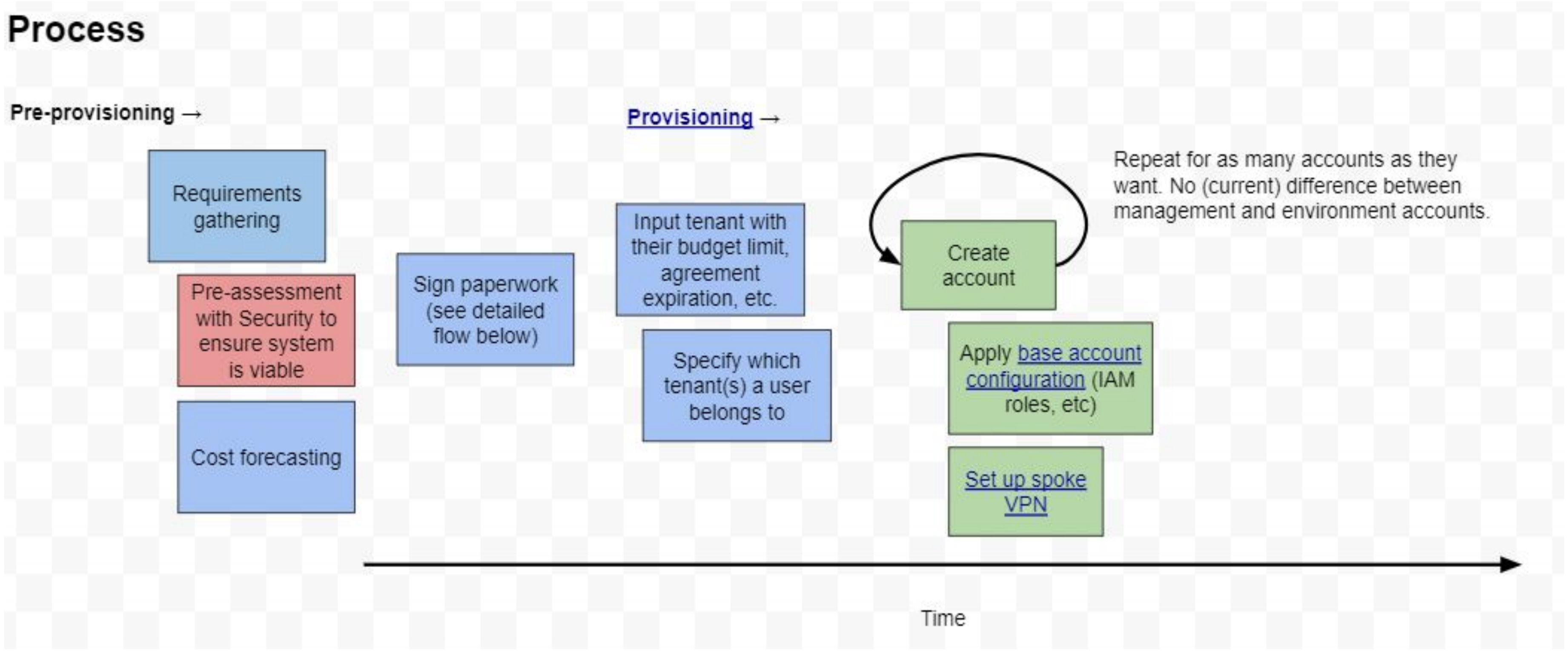
https://drive.google.com/drive/u/0/folders/1w8klCkjrrmci-e4_xhdNhKna7bIngS5U

GRACE High Level Architectural Details

GRACE Architecture Design considerations:

- One VPC for each environment and per AWS sub-account
  - Only one VPC per account (only one environment per account)
  - Each VPC per account will have a Virtual Private Gateway (VGW) configured with two tunnels for a VPN connection (via ASA router) back to NetOps VPC
  - The VGW will be attached to the VPC in that AWS account
- Strong isolation across environments. Role separation is easy to manage (security consideration). We don't have to worry to limit user to specific VPC's.
- Reduced risk of hitting AWS account level limits (eg: Each AWS account has a limit of 5 elastic IP's per region).
- Cost and billing management is managed at account level. With one-on-one mapping, application teams can review their costs per environment
- Better user management by limiting a user to a specific environment. User access controls can be handled optimally without posing risk to other environment VPC's (security is easy to manage).

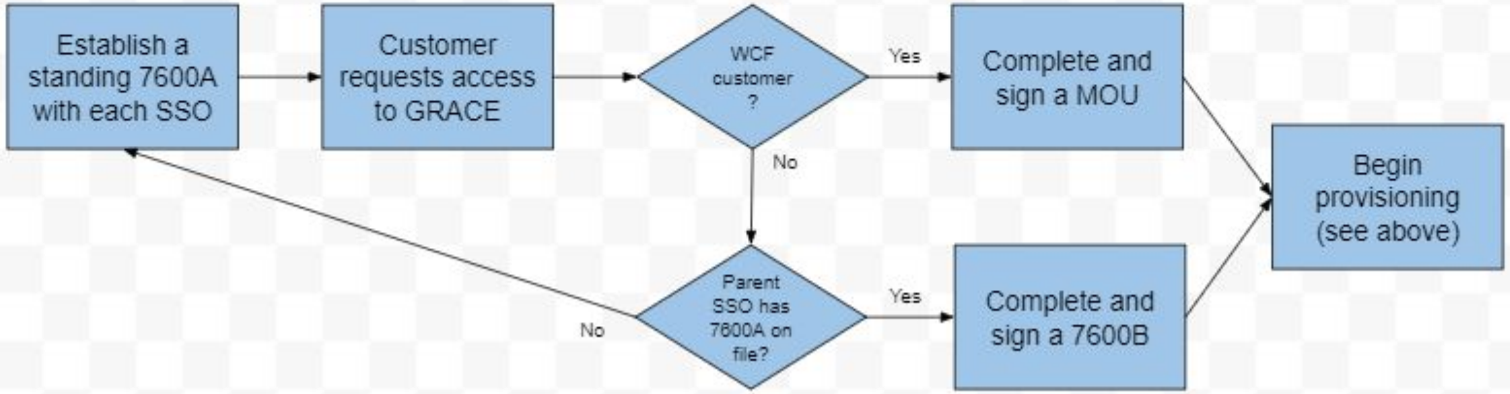
GRACE Account Provisioning Workflow


Process
Pre-provisioning →
Provisioning →
Requirements gathering
Pre-assessment with Security to ensure system is viable
Cost forecasting
Sign paperwork (see detailed flow below)
Input tenant with their budget limit, agreement expiration, etc.
Specify which tenant(s) a user belongs to
Create account
Repeat for as many accounts as they want. No (current) difference between management and environment accounts.
Apply base account configuration (IAM roles, etc)
Set up spoke VPN
Time

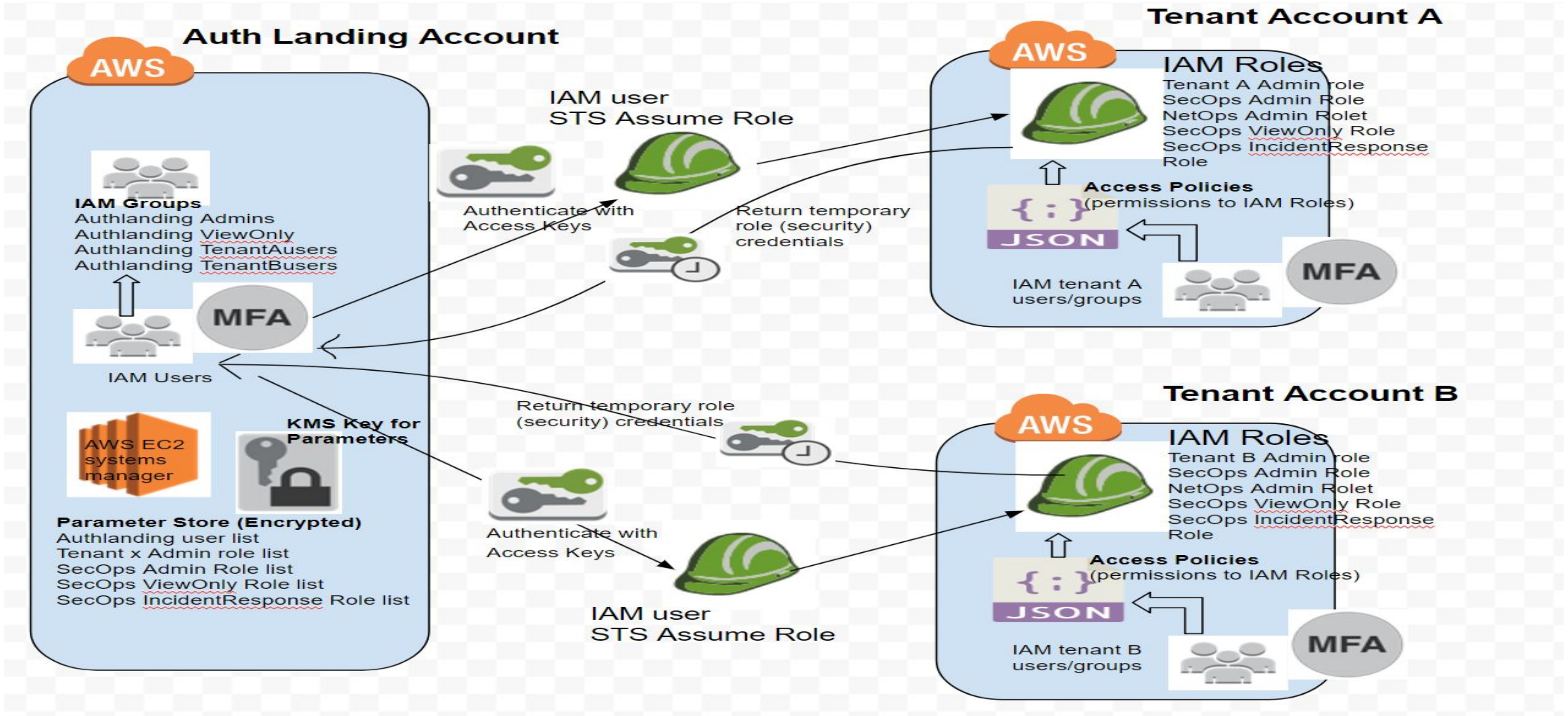
GRACE Account Provisioning Workflow...contd

Detailed flow for "Sign paperwork" step above →

See also: IAA/MOU Overview


Establish a standing 7600A with each SSO
Customer requests access to GRACE
WCF customer ?
Yes
Complete and sign a MOU
No
Parent SSO has 7600A on file?
No
Yes
Complete and sign a 7600B
Begin provisioning (see above)

GRACE User Provisioning Workflow



GRACE Incremental LATO Process

- GRACE Platform will undergo LATO approval process (30 security controls)
- Components can be assessed individually
    - Component level LATO approval process
    - List of Platform level LATO Components
- The assessment will include approval of the code (IaC) to provision various infrastructure components in tenant VPC's as well
- Pre-built "security Groups" and "NACL's" (via code)
    - If tenant's application can function with the pre-built SG's and NACL's (via code), the timeline for the LATO process of the tenant application will be significantly reduced

GRACE Standard Offering

- Onboarding
  - Tenant's self-managed AWS sub account(s)
  - Funding amount assigned to the tenant account (from IAA)
  - Tenant IAM admin account provisioned
  - Service Control Policy (SCP) attached to the tenant account to access all ISE approved AWS service offerings
  - Each tenant will have their own mgmt VPC and environment(s) VPC's
- Permissions to GRACE GitHub repos to access standard scripts/code offerings (IaC) (see below)
- Networking
  - Transit VPC - Security controls for egress and ingress network traffic
  - Connectivity to proxy server
  - Connectivity to GSA on-prem
  - Connectivity to SecOps VPC (to send logs)
  - VPN connectivity from tenant's mgmt VPC to tenant's environment VPC's
- Access to security approved hardened AMI's (Linux and Windows)
- Access to Ansible playbook to install/configure security agents (Nessus, OSSEC (HIDS), AV)

GRACE Servicing Agency (GSA IT-IDI) Responsibilities

- Support the Requesting (Tenant) Agency
- Provision networks, Transit VPC Environments and Provision IAM accounts
- Provide platform support services
- Provide cloud infrastructure SMEs for support issues and resolutions, as required
- Provide AWS Product & Services, AWS Marketplace via VAR
- AWS Service Commitments & SLAs (inherent)
- Compliant to Authority to Operate (ATO) for underlying IaaS
- Provide Operational Level Agreements and/or Service Level Agreements

GRACE Requestor (Tenant’s) Responsibilities

- Tenant’s requirements template completed
- Security
  - Tenant obtain their own LATO/ATO at the tenant level
  - Tenant is responsible for documenting and describing in detail the inbound/outbound policies (connection requirements) for the Application interconnections external to GSA
  - Tenant maintain an asset inventory for all core infrastructure and management components to include: virtual network assets such as security groups, subnets, VPN's, ACL's, and other similar asset components
  - Tenant capture all audit records IAW GSA policy for Database, Application, and Web Application components
- Financials
  - IAA or MOU
  - Tenant manage their cost/billing of all their components in their application tenant accounts
  - Creation of ROM (w/IDI Collaboration)
- GRACE tenant should use the standard scripts provided in GRACE GitHub repo to launch/provision infrastructure (IaC)
- Tenant is responsible to open/manage their AWS support tickets pertaining to changes to sub-account limits and relevant support requirements for all their application tenant accounts
- Additional in-depth tenant responsibilities documented here

GRACE Leveraged GSA Services

- Vulnerability Scanning
  - Nessus - Scans for security vulnerabilities, assures security compliance
- Enterprise Logging (OSSEC)
  - Satisfies security controls for log retention, audit and security incident forensics
- AntiVirus Engine
  - ClamAV (open source)
  - Cylance (SaaS solution)

GRACE Components (majority are open source products)

- Amazon Web Services (AWS)
- AWS Console
- Git
- GitHub
- Terraform
- CircleCI
- Ansible
- Slack
- Trello

- Specific AWS Services
- Leveraged GSA IT Services
- GitHub Repositories
- Cloud Management Platform
  - CloudCheckr (pending)

Amazon Web Services

Description

Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis. The technology allows subscribers to have at their disposal a full-fledged virtual cluster of computers, available all the time, through the Internet.

Purpose

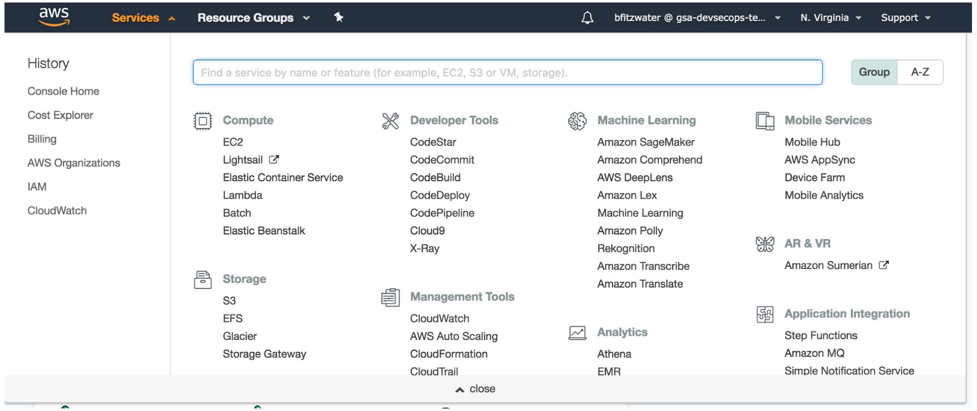
- **Infrastructure as a Service (IaaS)**
- **Current focus for infrastructure to support GRACE platform, though other cloud providers may be considered in the future**

AWS Management Console

Description
The AWS Management Console provides a simple and intuitive web-based user interface to access and manage Amazon Web Services. You can also use the AWS Console mobile app to quickly view resources on the go.

Purpose

- **Verify success of terraform actions**
- **Monitor services via Cloud Watch**
- **Monitor billing**
- **Perform actions not possible with terraform**
- **Circumvent philosophy of Infrastructure as Code**



Git

Description

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

Purpose

- **Software Configuration Control**
- **Client for Github**

GitHub

Description
GitHub is a web-based hosting service for version control using git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

Purpose

- **Software Configuration Management**
- **Infrastructure as Code governance**
- **Repository for all GRACE software and configuration components**
- **Issue tracking with integration to Trello**

Terraform

Description
Terraform is an infrastructure as code software by HashiCorp. It allows users to define a datacenter infrastructure in a high-level configuration language, from which it can create an execution plan to build the infrastructure such as OpenStack or in a service provider such as IBM Cloud (formerly Bluemix), AWS, Microsoft Azure or Google Cloud Platform. Infrastructure is defined in a HCL Terraform syntax or JSON format.

Purpose
- **Infrastructure as Code**
- **Fundamental building block of GRACE platform**

CircleCI

Description
CircleCI is a continuous integration and delivery platform. It is offered as both SaaS or as software you can install on your own infrastructure. It makes it easy for teams of all sizes to rapidly build and release software at scale with automated build, unit and integration testing.. Supports builds on Linux, macOS, and Android.

Purpose
- **Continuous Integration/Continuous Deployment (CI/CD)**
- **Automated build testing**

Ansible

Description
Ansible is software that automates software provisioning, configuration management, and application deployment.

Purpose

- **Configuration Management/Automation**
- **Configuration as Code**
- **OS Hardening**
- **Software Deployment**

Slack

Description

Slack is a cloud-based set of proprietary team collaboration tools and services, founded by Stewart Butterfield. Slack began as an internal tool used by their company, Tiny Speck, in the development of Glitch, a now defunct online game. The name is an acronym for "Searchable Log of All Conversation and Knowledge".

Purpose

- **Team communication and coordination**
- **Integration with GitHub and Trello facilitates team awareness of activities**
- **Integrations with CircleCI, Splunk, CloudWatch, etc. enables “ChatOps”**

Trello

Description
Trello is a web-based project management application. It employs a “kanban” model to show tasks as “cards” on “boards” arranged in lists or columns.

Purpose
- **Kanban**

Specific AWS Services

- Simple Storage Service (S3)
- Elastic Cloud Compute (EC2)
- Elastic Block Store (EBS)
- Organizations
- Application Load Balancers (ALB)
- Network Load Balancers (NLB)
- Key Management Service (KMS)
- Relational Database Service (RDS)
- Identity & Access Management (IAM)
- Other GSA security (ISE) approved AWS services

GRACE GitHub Repositories

- GSA/*DevSecOps*
- GSA/*devsecops*-example
- GSA/security-benchmarks
- GSA/*devsecops*-cloud-custodian-rules
- GSA/*devsecops*-tenant-networking
- GSA/*devsecops*-log-forwarding
- GSA/*devsecops*-ekk-stack
- GSA/ansible-fluentd
- GSA/*devsecops*-iam-roles
- GSA/*devsecops*-ebs-backup
- GSA/*devsecops*-subaccount-admin
- GSA/*aws-account-broker*

GRACE GitHub Repositories...Cont'd

- GSA/*DevSecOps*
  - Documentation only repository with information on DevSecOps and GRACE platform
- GSA/*devsecops*-example
  - Example terraform, ansible and packer configurations to setup VPC, networking, EC2 to deploy a Wordpress instance in the GRACE platform
- GSA/security-benchmarks
  - Ansible roles for hardening RHEL and Ubuntu instances
  - Python script for creating AWS Service Control Policy (SCP) from a CSV export of the AWS service approval tracking spreadsheet.
- GSA/*devsecops*-cloud-custodian-rules
  - Start on some Cloud Custodian rules for GRACE platform

GRACE GitHub Repositories...Cont'd

- GSA/*devsecops*-tenant-networking
    - Terraform code for a test implementation of the GRACE tenant networking model
- GSA/*devsecops*-log-forwarding
    - Terraform module creates infrastructure for collecting and forwarding logs including an autoscaling fluentd cluster in AWS
- GSA/*devsecops*-ekk-stack
    - Terraform code to create an Amazon Elasticsearch Service, Amazon Kinesis, and Kibana (EKK) stack for log aggregation and analysis
- GSA/ansible-fluentd
    - Ansible roles to install fluentd for system log parsing and streaming

GRACE GitHub Repositories...Cont'd

- GSA/*devsecops*-iam-roles
  - Terraform code to set up IAM roles and policies for the GRACE platform
- GSA/*devsecops*-ebs-backup
  - Quick implementation of an EBS backup solution with Terraform and AWS Lambda
- GSA/*devsecops*-subaccount-admin
  - Instructions and Ruby code for setting up AWS CLI and web console for subaccount administrator access.
- GSA/*aws-account-broker*
  - Account broker for creating Amazon organization accounts. Deprecated. Will use Terraform.

GRACE Points of Contact

- **Executive Sponsors**
  - Navin Vembar (CTO)
  - Brian Muolo (IDI)
- **Product Owner**
  - Electra Holmes (IDI)
- **Project Scrum Master**
  - Lulit Tesfaye (CTO Office)
- **Core Team Members**
  - Aidan Feldman (18F)
  - Manoj Chalise (ISE)
  - Jason Miller (IDI)
  - Brian Fitzwater (IDI)
  - Jeff Fredrickson (CTO Office)
  - Kishore Kakani (CTO Office)

GSA IT

GRACE Cloud Platform

$\underbrace{\text{GRACE}}$ ⬇

GSA Readily Available Computing Environment

Questions ?