Sce team(experience,claims )work(provisioning of application)based on infrastructure with pipe lines

Another team code reviews

IAC and aws we check

Sree kanth and arun (platform):-aws account,platform and tfe and ADHOC requests ( that is work on different cards)

3 accounts and 4 envitonment(dev,test,prod)(dev,stage,test,prod)

Opa policies according to erie standards

**Terraform enter pripise**

Opa policies(open policy agent)

State files

Terraform registry

**For git lab**

Cicd piplines and repo

Gitlab will store projects and workspace what update will do and then aws will act

Make an ec2 instance and push in git lab

Work :- create an ec2 instance and push to gitlab and write a yaml that is pipeline to commit and connect thius git lab to aws and crate an instance in aws with the help of gitlab and pipeline

**Branch:-**

Main branch the another branches will have copy of that so main branch will not disturbed…

**Process:-**

1)Mr 2)plan 3)approver and assigne 4)apply

Difference between 1.0 and 20.0 htat is portfolio and environment

Archetype it is blueprint done by ccoe

**2 types o architect usin now**

**1)ecs:-agent locator**

**2) lambda: using anehl**

**Trunk based feature based**

**Modules:-**

In Terraform, the aws\_caller\_identity **data source** is used to **retrieve information about the IAM identity**

**Local and data**

**Local:-if we want to do change any functions**

**Data:-read only**

**Roles according to ecs**

**1)container role**

**2)execution role**

**Normal role:-**

**1)execution role**

**2\_IAM Role**

**Logging:-where we get error we can see**

**10 info trace warning error**

**Agency application**

**Directory structure**

**Data.tf auto.tfvars main.tf**

**We cannot keep tags for data and locals**

**Based on your notes, here are the key topics covered:**

**✅ Teams and Responsibilities**

* **SCE Team (Experience/Claims): Application provisioning using infrastructure and pipelines.**
* **Platform Team (Sreekanth & Arun): Manages AWS accounts, Terraform Enterprise (TFE), and platform setup.**
* **Code Review Team: Handles MR reviews and approvals.**

**✅ Infrastructure Setup**

* **AWS Environments: Multiple accounts and four environments (dev, stage, test, prod).**
* **Terraform Enterprise (TFE): Used for IAC deployments.**
* **OPA Policies: Enforced according to Erie standards.**
* **State Files: Managed and stored securely.**

**✅ GitLab Integration**

* **GitLab Repos & Workspaces: Store code/projects and manage CI/CD.**
* **EC2 Automation:**
  + **Create EC2 using Terraform.**
  + **Push code to GitLab.**
  + **Configure .gitlab-ci.yml pipeline.**
  + **Use GitLab to apply infrastructure changes to AWS.**
* **Branches:**
  + **Feature branches copied from main.**
  + **Merge Request (MR) workflow: Plan → Approver → Apply.**

**✅ Terraform Concepts**

* **Modules: Reusable infrastructure components.**
* **Archetype: Blueprint provided by CCOE (Center of Cloud Excellence).**
* **Local vs Data:**
  + **local: Used for internal logic/calculations.**
  + **data: Read-only external resource fetch.**
* **aws\_caller\_identity: Data source to fetch IAM identity.**
* **File Structure:**
  + **main.tf, data.tf, auto.tfvars**
  + **Tags can't be applied to locals or data.**

**✅ ECS & Lambda**

* **ECS: Used with Agent Locator.**
* **Lambda: Used with Anehl (likely a custom internal tool or service).**
* **Roles:**
  + **ECS Roles: Container Role, Execution Role.**
  + **General IAM Roles: Execution Role and IAM Role.**

**✅ CI/CD Workflows**

* **Trunk-Based and Feature-Based development supported.**
* **Pipeline Stages:**
  + **Plan**
  + **Approve**
  + **Apply**

**✅ Logging Levels**

* **INFO, TRACE, WARNING, ERROR**

**Let me know if you want me to create a diagram or expand any specific section like ECS roles, pipeline stages, or GitLab structure.**

**Products services corporate enablement 249 applications**

**Corporate enablement data portfolioand application**

**Services to watch:-**

**Code commit build deploy**

**Nw:-vpc,route53**

**Compute:-ec2,lambda**

**Containers:-ecs ecr**

**Sec:-iam,kms,secrets manager,aft**

**When we create a control tower then it will create a account factory as service catolog product and create account**

**Suppose in organization u have multiple team u can create multiple organization unit in this ou u can create multiple accounts for example we have ou1 and ou2 and in that ou1 naad ou2 we can ceate acc1 and acc2 these two accounts are created through account factory. So if u use service catalog product that help to create account\**

**You're absolutely right — here's a clearer breakdown and explanation of what you're describing regarding AWS Control Tower, Account Factory, and Organizational Units (OUs):**

**✅ How AWS Control Tower Works (with OU and Account Factory)**

**1. Organizational Units (OUs):**

* **In AWS Organizations, you can group accounts under OUs like OU1, OU2, etc.**
* **Each OU can have multiple AWS accounts.**
* **This helps apply policies (guardrails) and manage accounts based on team or function (e.g., Dev, QA, Prod).**

**2. Account Factory (as Service Catalog Product):**

* **When you enable Control Tower, it automatically sets up an Account Factory using AWS Service Catalog.**
* **This is a self-service tool that allows users to provision new accounts into specific OUs with predefined configurations (networking, security baselines, etc.).**
* **You can:**
  + **Choose the OU where the account will reside (e.g., OU1 or OU2)**
  + **Provide details like account name, email, IAM role name, etc.**

**🧱 Example:**

**You have:**

| **OU Name** | **Accounts Created** |
| --- | --- |
| **OU1** | **account1, account2** |
| **OU2** | **account3, account4** |

**All accounts are created via Account Factory (Service Catalog Product).**

**This ensures:**

* **Consistent baseline configuration**
* **Integration with Control Tower guardrails**
* **Centralized auditing and logging via Control Tower setup**

**✅ Summary:**

| **Component** | **Purpose** |
| --- | --- |
| **Control Tower** | **Manages multi-account AWS setup with governance** |
| **Organizational Unit (OU)** | **Groups accounts logically (e.g., per team or environment)** |
| **Account Factory** | **Provisioning tool (Service Catalog product) to create accounts** |
| **Service Catalog** | **Backend service used to define and launch account templates** |

**Not exactly — but you're close! Let’s break it down clearly to avoid confusion:**

**✅ What is AFT (Account Factory for Terraform)?**

**AFT is NOT a Service Catalog product itself, but it works with Control Tower and Service Catalog to automate AWS account creation and customization using Terraform.**

**🔧 AFT = Account Factory for Terraform**

**It is:**

* **An open-source AWS solution provided by AWS.**
* **Built on top of Control Tower and Account Factory.**
* **Uses Terraform + Git repositories to automate account provisioning.**
* **Great for scale — when you need to create/manage many accounts automatically and consistently.**

**🔁 Difference between Account Factory and AFT**

| **Feature** | **Account Factory** | **AFT (Account Factory for Terraform)** |
| --- | --- | --- |
| **How it's used** | **Manually via AWS Console (Service Catalog product)** | **Automatically using Git + Terraform** |
| **Customization** | **Limited (GUI-based inputs)** | **Highly customizable (code-based)** |
| **Automation** | **Manual steps** | **Full automation (CI/CD-style account creation)** |
| **Use Case** | **Few accounts** | **Many accounts at scale, with custom setup** |
| **Trigger** | **Human triggers creation** | **Git commit triggers account creation** |

**🧠 Does AFT help during high traffic?**

**No, not directly.**

**AFT is not a load balancer or autoscaler. It is used to provision new AWS accounts — not to respond to high traffic on websites or services.**

**However, it helps with:**

* **Creating new accounts automatically, e.g., when new teams or projects are onboarded.**
* **Ensuring each account follows security, networking, and tagging standards.**
* **Integrating Terraform modules for infrastructure setup per account.**

**✅ Real-world Example:**

**Imagine your company is growing, and you want:**

* **A new AWS account per team (Dev, QA, Prod).**
* **Each account must have logging, networking, IAM roles pre-configured.**

**With AFT, you:**

1. **Push a config (e.g., JSON/Terraform) to a Git repo.**
2. **AFT picks it up via pipeline.**
3. **A new account is created with all settings in place — automatically.**

**Let me know if you want a diagram or a Terraform example of how AFT works!**

**Great question — and no, AFT (Account Factory for Terraform) is not limited to just creating AWS accounts. It does much more.**

**✅ AFT does 3 main things:**

| **🔧 Function** | **💬 What it does** |
| --- | --- |
| **1. Creates AWS Accounts** | **Automates account creation using Control Tower’s Account Factory** |
| **2. Customizes New Accounts** | **Deploys Terraform modules automatically to set up VPCs, CloudTrail, IAM roles, etc. in newly created accounts** |
| **3. Manages Lifecycle Changes** | **Re-applies or updates infrastructure (networking, logging, policies, etc.) to existing accounts when code changes** |

**📌 So, AFT is used for:**

* **Automating multi-account provisioning**
* **Applying organization-wide standards (networking, tagging, security, etc.)**
* **Maintaining infrastructure consistency across all Control Tower-managed accounts**
* **Making Git-driven changes to new and existing accounts**

**📘 Example:**

**You define a Terraform module for:**

* **VPC**
* **Security Groups**
* **S3 logging bucket**

**Whenever a new account is created, AFT:**

1. **Creates the account (via Account Factory)**
2. **Automatically runs Terraform to apply your VPC, security, and S3 bucket setup**

**🔁 Bonus: It’s GitOps-style**

**Everything runs based on changes to Git repos, so you don’t have to trigger things manually — just update code!**

**Would you like a real AFT architecture diagram or repo layout next?**