
PROGRAM SYLLABUS

AWS Cloud Engineer / Architect Program

Learn AWS by taking **i27CoreHR**, a real product, from an empty AWS account to live.

LIVE, INSTRUCTOR-LED**REAL PRODUCT IMPLEMENTATION****BASICS TO ADVANCED****21**

AWS modules, in the order the product needs them

1

Real product, i27CoreHR

2

Tracks, with or without Terraform

Live

Classes on your own AWS account

WHAT YOU BUILD ON AWS**Security & access**

IAM, Secrets Manager, WAF

Networking & delivery

VPC, Route 53, CloudFront

Compute & scaling

EC2, ALB, Auto Scaling

Storage & databases

S3, EBS, RDS, Aurora

Containers & serverless

ECS, Fargate, EKS, Lambda

Monitoring & auditing

CloudWatch, CloudTrail, Config

HOW THIS PROGRAM WORKS

You do not study AWS services. You implement them on a real product.

From day one you work as the cloud engineer for **i27CoreHR**, a real product that i27Academy runs. Every AWS service is taught at the point the product needs it, and you implement it yourself on your own AWS account. That is how cloud engineers learn on the job, and it is the experience this program gives you: **not just knowing AWS, but having built, secured, scaled, and fixed a real system on it.**

A TYPICAL AWS COURSE	THIS PROGRAM
Learn each service on its own	Learn each service because the product needs it
Spin up a demo, then delete it	Build on what you built yesterday. It keeps running
Only the happy path	Break it, troubleshoot it, fix it
Know what a service is	Explain why it is there and what you would change

The project flow: i27CoreHR, from your laptop to live

21 AWS modules · Terraform in Track 2

- 1 Understand the product and run it locally**
Architecture, source code, application and database
MODULE 01
- 2 Set up and secure the AWS account**
Account security, budgets, CLI, Regions, IAM
MODULES 02 – 04
- 3 Build the network and deploy the product**
VPC, EC2, Systems Manager, Secrets Manager, EBS, EFS
MODULES 05 – 08
- 4 Scale it and put it on a domain**
Launch templates, Auto Scaling, Load Balancing, WAF, Route 53, HTTPS, S3
MODULES 09 – 13
- 5 Move data to managed services**
Amazon RDS and Aurora
MODULE 14
- 6 Containerize how it ships**
Docker, ECR, ECS, Fargate, EKS
MODULES 15 – 18
- 7 Add serverless and messaging**
Lambda, API Gateway, SQS, SNS
MODULE 19
- 8 Connect to networks outside AWS**
Site-to-Site VPN, Direct Connect
MODULE 20
- 9 Monitor and audit it in production**
CloudWatch, CloudTrail, AWS Config
MODULE 21
- 10 Build infrastructure as code** **TRACK 2**
Terraform, applied to a second real product, i27Helpdesk
20 TERRAFORM MODULES · 7 PROJECT PHASES

The result: i27CoreHR, running on AWS

Every layer built, secured, scaled, and monitored by you, the way a cloud engineer would.

LIVE

STEP 01

Understand the product and run it locally

Before anything moves to AWS, get to know i27CoreHR the way a new engineer on the team would.

01 i27CoreHR · Product introduction & local implementation

Why it matters · You cannot run a product on AWS until you know how it runs at all.

- Introduction to i27CoreHR
- Understanding the application and database connection
- Getting the source code
- Building and running the application locally
- Reading logs and troubleshooting startup issues
- Application architecture and technology stack
- Installing the required local tools
- Configuring the application and database
- Validating application functionality and data persistence
- *and more*

Applied to i27CoreHR · The product runs on your own machine, connected to its database, ready to be taken to AWS.

STEP 02

Set up and secure the AWS account

An empty AWS account, set up the way an organization would before anything is deployed in it.

02 AWS Account Setup · Account security, budgets & CLI setup

Why it matters · Everything you build lives in this account, so it is secured and cost-controlled first.

- Creating an AWS account
- Securing the root account with MFA
- Exploring the AWS Management Console
- Understanding AWS Free Tier and billing
- Installing the AWS CLI
- *and more*
- Understanding the root user
- Setting up administrative access
- Selecting an AWS Region
- Setting up AWS Budgets and cost alerts
- Configuring CLI access and verifying connectivity

Applied to i27CoreHR · The account i27CoreHR will run in is secured, has a budget with alerts, and is reachable from your terminal.

03 Cloud Foundations · Regions, Availability Zones & shared responsibility

Why it matters · Every AWS design decision you make later rests on these ideas.

• CLOUD COMPUTING

- Traditional infrastructure vs cloud
- Public, private, and hybrid cloud
- IaaS, PaaS, and SaaS

• AWS GLOBAL INFRASTRUCTURE

- Regions
- Availability Zones
- Edge locations

• CORE CLOUD CONCEPTS

- Compute, storage, networking, and databases
- Scalability, elasticity, and high availability
- Shared responsibility model
- Pay-as-you-go pricing

Applied to i27CoreHR · See what i27CoreHR needs from the cloud, where it will run, and which parts of it are yours to secure.

04 Identity & Access Management · IAM

Why it matters · Every person and every service that touches i27CoreHR gets its access through IAM.

- IAM users and groups
- MFA and access keys
- Policy structure and ARNs
- Temporary credentials
- Credential reports
- IAM Access Analyzer
- Console and programmatic access
- IAM policies and permissions
- IAM roles and trust policies
- Least-privilege access
- Access Advisor
- *and more*

Applied to i27CoreHR · Create the users, groups, and roles i27CoreHR's team and servers need, each with least-privilege permissions.

STEP 03

Build the network and deploy the product

Design the private network i27CoreHR lives in, then put the product on servers inside it.

05 Networking · Amazon VPC

Why it matters · The network decides what is public, what stays private, and how traffic reaches the product.

- VPC and CIDR blocks
- Route tables and routing
- NAT Gateway
- Network ACLs (NACLs)
- Bastion host and private instance access
- VPC endpoints
- Network connectivity and troubleshooting
- Public and private subnets
- Internet Gateway
- Security groups
- Security groups vs NACLs
- VPC peering
- VPC Flow Logs
- *and more*

Applied to i27CoreHR · Build the VPC i27CoreHR runs in, with public and private subnets, routing, and firewall rules for each layer.

06 Compute · Amazon EC2 & i27CoreHR deployment

Why it matters · This is where i27CoreHR first runs on AWS, on servers you launch and configure yourself.

- EC2 instances and instance types
- Launching and connecting to instances
- Private and public IP addresses
- Application installation and deployment
- Managing applications with systemd
- Instance lifecycle and troubleshooting
- Amazon Machine Images (AMIs)
- Key pairs and SSH access
- Elastic IP addresses
- User data and startup automation
- Creating and using custom AMIs
- *and more*

Applied to i27CoreHR · Deploy i27CoreHR on EC2 inside your VPC, start it automatically on boot, and capture it as a custom AMI.

07 Systems Manager & Secrets Manager

Why it matters · Real teams manage servers without opening SSH to the internet, and never keep passwords in code or config files.

- Systems Manager overview
 - Session Manager
 - Parameter Store and parameter types
 - Parameter Store vs Secrets Manager
 - Secret rotation basics
 - SSM Agent and IAM permissions
 - Run Command
 - Creating and retrieving secrets
 - IAM permissions and KMS encryption
 - VPC endpoints and troubleshooting
- *and more*

Applied to i27CoreHR · Reach i27CoreHR's private servers through Session Manager, and move its configuration and database credentials into Parameter Store and Secrets Manager.

08 Storage & Backups · Amazon EBS & EFS

Why it matters · Servers get replaced, but the product's data has to survive, be shared, and be restorable.

- EBS volumes and volume types
 - Formatting and mounting volumes
 - Expanding volumes and file systems
 - EBS snapshots and restoration
 - EFS concepts and shared storage
 - Mounting EFS across multiple EC2 instances
 - Storage troubleshooting
 - Creating and attaching volumes
 - Persistent mounts across reboots
 - EBS encryption
 - Amazon Data Lifecycle Manager (DLM)
 - Creating EFS file systems and mount targets
 - EBS vs EFS
- *and more*

Applied to i27CoreHR · Give i27CoreHR's servers encrypted volumes, shared storage across instances, and scheduled snapshots you can restore from.

STEP 04

Scale it and put it on a domain

Turn one server into a setup that can be repeated, scaled with demand, and reached on a real domain.

09 Launch Templates

Why it matters · Scaling starts with being able to launch the same server, the same way, every time.

- Launch template concepts
 - Configuring AMIs and instance types
 - Storage and IAM instance profiles
 - Launching instances from templates
 - Updating templates for new application releases
 - Creating launch templates
 - Key pairs and security groups
 - User data and application startup
 - Template versions and default versions
- *and more*

Applied to i27CoreHR · Capture how an i27CoreHR server is built in a launch template, and roll out a new release as a new template version.

10

EC2 Auto Scaling

Why it matters · Traffic changes and servers fail. The product should add, remove, and replace servers on its own.

- Auto Scaling groups
 - Minimum, maximum, and desired capacity
 - Target tracking and step scaling
 - Health checks and instance replacement
 - Instance refresh and rolling updates
 - Launch template integration
 - Scaling across Availability Zones
 - Scheduled scaling
 - Instance warmup and cooldown
 - Scaling activities and troubleshooting
- o *and more*

Applied to i27CoreHR · Run i27CoreHR in an Auto Scaling group across Availability Zones, and roll out a new release with an instance refresh.

11

Load Balancing · Elastic Load Balancing & AWS WAF

Why it matters · Once there is more than one server, users need a single front door that sends them to a healthy one and keeps bad traffic out.

- Load balancing concepts
 - Network Load Balancer (NLB)
 - Listeners and target groups
 - Path-based and host-based routing
 - Auto Scaling group integration
 - Web ACLs and ALB integration
 - IP filtering and rate-based rules
 - WAF logs and troubleshooting
 - Application Load Balancer (ALB)
 - Internet-facing and internal load balancers
 - Health checks
 - HTTPS listeners and certificates
 - Traffic distribution and troubleshooting
 - AWS managed rules
 - Testing count and block actions
 - AWS Global Accelerator overview and ALB integration
- o *and more*

Applied to i27CoreHR · Put a load balancer in front of i27CoreHR's Auto Scaling group, with health checks and routing rules, and protect it with AWS WAF.

12

DNS & HTTPS · Amazon Route 53 & AWS Certificate Manager

Why it matters · A real product is reached by its own domain name, over HTTPS, not by a load balancer address.

- DNS concepts and domain names
 - DNS records and alias records
 - Route 53 routing policies
 - Mapping a domain to a load balancer
 - Requesting and validating certificates with ACM
 - Certificate renewal
 - Public and private hosted zones
 - Domain delegation and name servers
 - Health checks and DNS failover
 - SSL/TLS certificates
 - Configuring HTTPS on an ALB
 - DNS and HTTPS troubleshooting
- o *and more*

Applied to i27CoreHR · Put i27CoreHR on its own domain with Route 53 and serve it over HTTPS with a certificate from ACM.

13 Object Storage & Encryption · Amazon S3 & AWS KMS

Why it matters · Files, uploads, and backups do not belong on a server's disk. They need storage that is durable, access-controlled, and encrypted.

- S3 buckets and objects
- Storage classes
- Block Public Access
- Lifecycle rules
- Static website hosting
- Encryption at rest and in transit
- Key policies, aliases, and key rotation
- Access and storage troubleshooting
 - *and more*
- Uploading and downloading files
- Bucket policies and access permissions
- Object versioning and delete markers
- Presigned URLs
- S3 replication
- AWS KMS: AWS managed and customer managed keys
- S3 encryption with SSE-KMS
- CloudFront integration with S3 for content delivery

Applied to i27CoreHR · Store i27CoreHR's files in a private S3 bucket with versioning and lifecycle rules, encrypted with a KMS key.

STEP 05

Move data to managed services

Take the database off the server and run it on a managed service built for backups, failover, and scale.

14 Managed Databases · Amazon RDS & Aurora

Why it matters · The database holds what the business cannot lose, so it needs backups, failover, and private access.

- Managed database concepts
- Creating and connecting to an RDS database
- Public and private database access
- Backups, snapshots, and point-in-time recovery
- Read replicas
- Encryption with AWS KMS
- Database monitoring and troubleshooting
- RDS database engines
- DB subnet groups and security groups
- Parameter groups
- Multi-AZ deployments and failover
- Database storage and scaling
- Amazon Aurora
 - *and more*

Applied to i27CoreHR · Move i27CoreHR's database to a private, encrypted RDS instance with automated backups and Multi-AZ failover.

STEP 06

Containerize how it ships

Package i27CoreHR as container images, then run it on AWS's managed container platforms.

15 Containerization · Docker & containerizing i27CoreHR

Why it matters · Modern products ship as containers, so you learn Docker in depth and containerize the product yourself.

- Containers vs virtual machines
- Images, containers, layers, and tags
- Port mapping and container networking
- Writing Dockerfiles
- Multi-stage builds
- Container logs and troubleshooting
- Docker architecture and installation
- Running and managing containers
- Volumes and bind mounts
- Building and testing images locally
- Environment variables and application configuration
- *and more*

Applied to i27CoreHR · Containerize i27CoreHR: write its Dockerfile, build the image with a multi-stage build, and run the product as a container.

16 Container Registry · Amazon ECR

Why it matters · An image on your laptop is no use to AWS. It needs a private, scanned, versioned home that deployments pull from.

- ECR concepts and repositories
- Authentication and IAM permissions
- Pulling images from ECR
- Image vulnerability scanning
- Lifecycle policies and image cleanup
- *and more*
- Creating private repositories
- Tagging and pushing Docker images
- Image tag immutability
- Repository policies
- Authentication and image troubleshooting

Applied to i27CoreHR · Push i27CoreHR's image to a private ECR repository, with vulnerability scanning and lifecycle cleanup.

17 Container Deployment · Amazon ECS & AWS Fargate

Why it matters · Containers in production need something to run them, restart them, scale them, and roll out new versions safely.

- ECS concepts and architecture
- EC2 and Fargate launch types
- Container images from ECR
- Task execution roles and task roles
- Environment variables and secrets
- Service scaling
- CloudWatch logs and troubleshooting
- ECS clusters
- Task definitions, tasks, and services
- CPU and memory configuration
- Networking and security groups
- ALB integration and health checks
- Deployments and rollbacks
- *and more*

Applied to i27CoreHR · Run i27CoreHR as an ECS service on Fargate behind the load balancer, then deploy a new version and roll it back.

18 Managed Kubernetes · Amazon EKS

Why it matters · Many teams run containers on Kubernetes, so a cloud engineer should know how AWS offers it as a managed service.

- Kubernetes and EKS overview
- Creating an EKS cluster
- IAM access and cluster connectivity
- Pods, Deployments, and Services
- EKS control plane and worker nodes
- Managed node groups
- Configuring kubectl
 - *and more*

Scope · This module is for understanding the managed service. i27CoreHR is not deployed on EKS; it runs on ECS and Fargate.

STEP 07

Add serverless and messaging

Not everything needs a server. Run code on demand and connect parts of the system with queues and notifications.

19 Serverless & Messaging · Lambda, API Gateway, SQS & SNS

Why it matters · Some work is better done by a function that runs only when needed, and by parts that talk through messages.

- Serverless concepts
- Lambda execution roles
- API Gateway basics
- SQS queues and message processing
- Retries and dead-letter queues
 - *and more*
- Creating and invoking Lambda functions
- Environment variables, timeouts, and memory
- Creating APIs and integrating with Lambda
- SNS topics and subscriptions
- CloudWatch logs and troubleshooting

Applied to i27CoreHR · Build a serverless, event-driven flow alongside i27CoreHR: an API that triggers a function, a queue, and notifications.

STEP 08

Connect to networks outside AWS

How companies link their offices and data centers to their AWS network.

20 Hybrid Networking · Site-to-Site VPN & Direct Connect

Why it matters · Most companies still have offices and data centers, and those need a private path into AWS.

- Hybrid networking concepts
- Customer Gateway and Virtual Private Gateway
- Static and dynamic routing
- Direct Connect overview
 - *and more*
- Site-to-Site VPN architecture
- VPN tunnels
- VPN monitoring and troubleshooting
- VPN vs Direct Connect

STEP 09

Monitor and audit it in production

Know what the product is doing, get told when something is wrong, and keep a record of every change.

21 Monitoring & Auditing · CloudWatch, CloudTrail & AWS Config

Why it matters · A system you cannot see is a system you cannot fix. Running it in production means watching it and tracing every change.

- CloudWatch metrics and dashboards
 - Alarms and SNS notifications
 - Tracking AWS API activity
 - Basic compliance checks
 - Application logs and CloudWatch Agent
 - CloudTrail event history and trails
 - AWS Config and resource configuration history
 - Monitoring and troubleshooting resource changes
- o *and more*

Applied to i27CoreHR · Build dashboards and alarms for i27CoreHR, collect its application logs, and trace who changed what in the account.



TRACK 2 · ONLY IF YOU CHOOSE IT

Terraform and Infrastructure as Code

Everything up to this page is **Track 1**. The pages that follow are taught only in **Track 2**, where you learn Terraform and provision the AWS infrastructure for a second real product, **i27Helpdesk**, entirely as code.

YOUR ROLE IN TRACK 2

You are the **cloud engineer** responsible for provisioning the infrastructure that the DevOps team needs to deploy and operate i27Helpdesk.

YOU

Provision it with Terraform

Write the infrastructure as code, as reusable modules, and apply it to every environment.

WHAT YOU PROVISION ON AWS

i27Helpdesk's infrastructure

Everything the product needs to run on AWS, from the network up to the platform and data it runs on, entirely as code.

THE DEVOPS TEAM

Deploys and operates on it

Their pipelines release the product's 6 apps through every environment you built.

dev test stage prod

TRACK 1

2 months

AWS Cloud Engineer / Architect

- All the AWS modules before this page
- One real product: **i27CoreHR**
- Each AWS service implemented on the product

TRACK 2

3 months

AWS + Terraform

- **Everything in Track 1**, including i27CoreHR
- **Terraform**, from the ground up to advanced
- A second real product: **i27Helpdesk**, with its infrastructure 100% as code

TRACK 2

How to read the rest of this syllabus: every page and module that belongs to Track 2 carries this purple tag. If you are on Track 1, your program ends before this page.

TRACK 2 · TERRAFORM

Learn Terraform, build i27Helpdesk's infrastructure

The Terraform modules come first. The i27Helpdesk project flow on AWS follows them, phase by phase.

01 Introduction to IaC and Terraform

Why it matters · Before writing any code, understand the problem Infrastructure as Code solves and why teams choose Terraform.

- Challenges with traditional infrastructure
- Why and what is Terraform
- Course project overview
- Types of IaC: declarative vs imperative
- Terraform vs other IaC tools
- *and more*

02 Setup and Authentication

Why it matters · Terraform acts on your cloud account, so it has to be installed and signed in safely before anything else.

- Installing Terraform on Windows, macOS and Linux
- Local authentication with short-lived credentials
- IDE setup: VS Code with the Terraform extension
- Installing the cloud CLI
- Why to avoid long-lived access keys
- *and more*

03 Terraform Architecture and Workflow

Why it matters · Every Terraform change follows the same loop. Knowing it, and how to read a plan, is what keeps changes safe.

- Terraform architecture: core, providers, state
- Supporting commands: `fmt`, `validate`, `show`
- Local state: what `terraform.tfstate` stores
- *and more*
- Workflow: `init`, `plan`, `apply`, `destroy`
- Configuration syntax: blocks, arguments, `.tf` file structure
- Reading a plan: `create`, `update`, `replace`, `destroy`

04 Terraform Providers and Registry

Why it matters · Terraform knows nothing about any cloud on its own. Providers are how it talks to each one.

- What providers are
- Terraform Registry: official, partner and community providers
- Provider alias for multiple regions or accounts
- *and more*
- How Terraform finds and downloads providers
- Provider configuration
- Multiple providers in one configuration

05 Terraform Settings Block

Why it matters · Pinning Terraform and provider versions is what makes the same code behave the same way for everyone on the team.

- Purpose of the `terraform {}` block
- Required providers and version constraints: `=`, `>=`, `~>`
- Backend configuration (overview, covered in the State module)
- Required Terraform version
- Dependency lock file: `.terraform.lock.hcl`, upgrading with `init -upgrade`
- *and more*

06 Terraform Resource Block

Why it matters · Resources are the infrastructure itself. Everything else in Terraform exists to describe and connect them.

- What a resource block is
- Arguments vs attributes, and referencing attributes
- Resource behaviour: create, update in place, destroy and re-create, destroy
- *and more*
- Resource syntax: type, local name, arguments
- Implicit dependencies between resources
- Dependency graph and `terraform graph`

07 Terraform Data Sources

Why it matters · Not everything is created by your code. Data sources let you use what already exists without taking it over.

- What data sources are and when to use them
- Fetching existing infrastructure
- Data sources vs resources
- Data source syntax
- Combining data sources with resources
- *and more*

08 Terraform Input Variables

Why it matters · Variables turn one fixed configuration into code that can be reused across environments.

- What input variables are
- Variable types: string, number, bool, list, map, object
- Variable precedence
- Sensitive variables
- Defining variables: type, default, description
- Ways to provide values: defaults, prompts, `-var` and `-var-file`, `TF_VAR_name`, `terraform.tfvars` and `*.auto.tfvars`
- Validation rules
- *and more*

09

Terraform Output Values and Local Values

Why it matters · Outputs expose what you built to people and other code. Locals keep the configuration free of repetition.

- What output values are
- Sensitive outputs
- What local values are
- Variables vs locals vs outputs
- Defining outputs and accessing resource attributes
- `terraform output` command
- Using locals to reduce repetition
- *and more*

10

Terraform Meta-Arguments

Why it matters · Real infrastructure has many similar pieces. Meta-arguments create and control them without copy and paste.

- What meta-arguments are
- `for_each`: iterating over maps and sets, `each.key` and `each.value`
- `depends_on`: explicit dependencies
- `provider`: overriding the default provider
- `count`: creating multiple instances, `count.index`
- `count` vs `for_each`, and when to choose each
- `lifecycle`: `create_before_destroy`, `prevent_destroy`, `ignore_changes`, `replace_triggered_by`
- *and more*

11

Terraform Expressions and Dynamic Blocks

Why it matters · Expressions let the configuration make decisions and transform data, so the code adapts to its inputs.

- Types and values, string interpolation
- `for` expressions for lists and maps
- Dynamic blocks for repeated nested blocks
- *and more*
- Conditional expressions
- Splat expressions
- When not to use dynamic blocks

12

Terraform Functions

Why it matters · Built-in functions do the string, list, file, and network calculations you would otherwise hard-code.

- What functions are, testing them with `terraform console`
- Numeric functions: `min`, `max`
- Filesystem functions: `file`, `templatefile`, `filebase64`
- Type conversion and date/time functions: `tostring`, `tolist`, `toset`, `timestamp`
- String functions: `join`, `split`, `format`, `lower`, `replace`
- Collection functions: `length`, `concat`, `merge`, `lookup`, `flatten`
- IP network functions: `cidrsubnet`
- *and more*

13 Terraform Provisioners

Why it matters · Provisioners run scripts from Terraform. Knowing their limits tells you when to reach for something better.

- What provisioners are and why they are a last resort
- Connection block for remote access
- Failure behaviour with `on_failure`
- Better alternatives: VM bootstrap scripts and configuration management tools
- Types: `local-exec`, `remote-exec`, `file`
- Creation-time vs destroy-time provisioners
- `terraform_data` resource
- *and more*

14 Terraform State

Why it matters · State is Terraform's record of what it built. Teams can only work together once it is shared, locked, and protected.

- What state is and why Terraform needs it
- Remote backends and bootstrapping the state bucket
- State locking
- `terraform_remote_state` for sharing outputs between configurations
- *and more*
- Local vs remote state
- Migrating local state to a remote backend with `init -migrate-state`
- State commands: `list`, `show`, `mv`, `rm`, `pull`
- Protecting state: encryption, access control, versioning

15 Terraform Modules

Why it matters · Modules package infrastructure once so every environment and every team can reuse the same tested building blocks.

- What modules are: root and child modules
- Calling modules and passing variables between them
- Module versioning with Git tags and `?ref=`
- *and more*
- Creating a module: inputs, outputs, file structure
- Module sources: local, Git, Terraform Registry
- Module composition and best practices

16 Import and Refactoring

Why it matters · Real projects start with infrastructure that already exists, and code that has to be reorganized without rebuilding anything.

- Bringing existing resources under Terraform with the `import` block
- Generating configuration with `plan -generate-config-out`
- Removing resources from state with the `removed` block
- *and more*
- `terraform import` command vs `import` block
- Renaming and moving resources with the `moved` block
- Verifying refactors with a no-change plan

17 Workspaces and Environments

Why it matters · The same code has to run for dev, test, stage, and prod without one environment affecting another.

- What workspaces are
- Using `terraform.workspace` in configuration
- Folder-per-environment structure
- *and more*
- Workspace commands: `new`, `select`, `list`, `show`, `delete`
- Limitations of CLI workspaces
- Separate backends and variables per environment

18 Secrets Management

Why it matters · Passwords and keys leak easily through state and plan files. Handling them properly is part of writing Terraform.

- How secrets end up in state and plan files
- Ephemeral variables and ephemeral resources
- Integrating with a cloud secret manager
- *and more*
- Sensitive variables and outputs, and their limits
- Write-only arguments
- Best practices for handling secrets in Terraform

19 CI/CD Integration with GitHub Actions

Why it matters · On a team, infrastructure changes are reviewed and applied by a pipeline, not from someone's laptop.

- Why run Terraform through a pipeline
- Keyless authentication with OIDC
- Saved plans with `plan -out`, apply on merge
- *and more*
- GitHub Actions basics: workflows, jobs, steps
- `fmt`, `validate` and `plan` on pull requests
- Environment-based deployments and approval for production

20 Troubleshooting and Drift

Why it matters · Infrastructure gets changed outside Terraform and things break. Finding and fixing that is everyday work.

- What drift is and how it happens
- Resolving drift: update code or revert the change
- Common errors: state lock, provider version, dependency cycles
- *and more*
- Detecting drift with `plan -refresh-only`
- Debugging with `TF_LOG` and `TF_LOG_PATH`
- Recovering from state issues

TRACK 2 · PROJECT FLOW ON AWS

i27Helpdesk infrastructure project

Build the infrastructure the DevOps team needs to run i27Helpdesk, improving how it is managed at each phase.

01

Foundation

AFTER MODULE 09

- VPC, subnets, NAT Gateway, security groups, an EC2 instance, and an S3 bucket
- A single configuration with local state, using variables, outputs, and locals

02

Scale the configuration

AFTER MODULE 13

- Multiple subnets and security group rules with `for_each` and dynamic blocks
- IAM roles and policies, and an ECR container registry
- EC2 bootstrap script

03

Remote state

AFTER MODULE 14

- Bootstrap the S3 state bucket
- Migrate local state to the S3 backend, with locking

04

Local modules

AFTER MODULE 16

- Split the code into network, compute, IAM, registry, and storage modules
- Use moved blocks and confirm a no-change plan

05

Versioned modules and environments

AFTER MODULE 17

- Move the modules to a GitHub repository and consume them by tag
- Create dev, test, stage, and prod, each with separate state
- Release a new module version and upgrade one environment

06

Database and Kubernetes

AFTER MODULE 18

- RDS MySQL database, with its password handled securely
- EKS cluster and node group

07

CI/CD

AFTER MODULE 19

- GitHub Actions with keyless authentication: GitHub OIDC with an IAM role
- Plan on pull request, apply on merge, approval for prod

Handover. i27Helpdesk's infrastructure on AWS exists as versioned, reviewed code, ready for the DevOps team to deploy and operate the product on.

NEXT STEP

Learn AWS the way a cloud engineer works. On a real product.

Choose the track that fits you. Both are built around implementing real products, not watching demos.

TRACK 1

2 months

AWS Cloud Engineer / Architect

- **21 AWS modules**, basics to advanced
- One real product: **i27CoreHR**
- Each AWS service implemented on the product

TRACK 2

3 months

AWS + Terraform

- **Everything in Track 1**, including i27CoreHR
- **20 Terraform modules**, from the ground up to advanced
- A second real product: **i27Helpdesk**, with its infrastructure 100% as code

Live, instructor-led classes

Taught live, with time for questions in every session.

Daily class recordings

Every session is recorded and shared, so you can catch up or revise.

Your own AWS account

You build everything yourself, hands-on, not in a shared sandbox.

Real products

Production-grade products that i27Academy runs.

S

TAUGHT LIVE BY

Siva · Lead trainer, i27Academy

14+ years across software development, cloud, DevOps, SRE, and AI implementation.

TALK TO US

Join the next batch

WEBSITE

i27academy.com

CALL & WHATSAPP

+91 93810 62032