To meet the requirements outlined in the scenario, we'll design a Kubernetes-based platform using AWS (Amazon Web Services) infrastructure. We'll utilize Terraform to manage the infrastructure as code, and Kubernetes to orchestrate containerized applications. For automation and CI/CD pipelines, we'll use tools like Jenkins or GitLab CI/CD. Let's break down the design and create the necessary components:

1. **AWS Infrastructure Components:**

a. VPC (Virtual Private Cloud):

* + Create a VPC that spans multiple availability zones (AZs) to ensure high availability and fault tolerance.
  + Divide the VPC into subnets, such as public subnets for load balancers, and private subnets for backend services.

b. Security Groups:

* + Define security groups to enforce network-level isolation between products.
  + Allow communication within a product's services and restrict access from other products.

c. IAM (Identity and Access Management) Roles:

* + Set up IAM roles and policies to grant teams access to their respective resources only.
  + Ensure proper permissions for managing AWS resources.

d. AWS EKS (Elastic Kubernetes Service):

* + Create an EKS cluster to manage the Kubernetes control plane.
  + Utilize multiple node groups across different AZs for scaling and resiliency.
  + Set up auto-scaling for worker nodes to handle spikes in load.

e. AWS RDS (Relational Database Service):

* + Offer managed databases to products that require relational data storage.
  + Isolate databases by creating them within the appropriate VPC and subnet.

f. AWS S3 (Simple Storage Service):

* + Use S3 for object storage, especially for teams requiring data storage.
  + Implement access controls to ensure isolation between products.

g. AWS ELB/NLB (Elastic Load Balancer/Network Load Balancer):

* + Use ELB/NLB to distribute traffic to services within products.
  + Utilize target groups for dynamic service discovery.

h. Route 53:

* + Use Route 53 for DNS management.
  + Automate DNS record creation for new products and services using CLI or SDKs.

1. **Kubernetes Cluster Configuration:**

a. Namespace Per Product:

* + Create a Kubernetes namespace for each product to ensure logical isolation.
  + Implement Role-Based Access Control (RBAC) to grant permissions within namespaces.

b. Resource Quotas:

* + Set up resource quotas for each namespace to prevent resource hogging by a single product.

c. Ingress Controller:

* + Deploy an Ingress controller (e.g., Nginx Ingress Controller) to manage external access to services.
  + Leverage Ingress resources for routing external traffic to products.

1. **CI/CD Pipeline:**

a. Git Repository for Kubernetes Manifests:

* + Create a central Git repository to store Kubernetes manifests for all products.
  + Use a folder structure to organize manifests by products.

b. Jenkins or GitLab CI/CD:

* + Set up CI/CD pipelines triggered by changes to the Kubernetes manifests.
  + Automate the deployment of new products or updates to existing products.

1. **Observability and Monitoring:**

a. Prometheus and Grafana:

* + Deploy Prometheus for collecting metrics from Kubernetes and applications.
  + Set up Grafana to visualize and monitor the cluster's health.

b. ELK Stack (Elasticsearch, Logstash, Kibana):

* + Centralize logging with the ELK stack to enable team members to view logs.

1. **Documentation:**

a. System Overview:

* + Document the architecture and components of the platform.
  + Explain how the different AWS services and Kubernetes components interact.

b. Deployment Instructions:

* + Provide step-by-step guides on deploying new products and services.
  + Include instructions for setting up CI/CD pipelines.

c. Access and Security:

* + Explain how teams can access their products securely.
  + Detail IAM roles, security groups, and namespaces' access controls.

d. Observability and Monitoring:

* + Provide instructions on accessing and using Prometheus, Grafana, and ELK Stack.
  + Offer best practices for monitoring applications.

e. Scaling and High Availability:

* + Explain how the platform handles spikes in load and scales resources.
  + Describe the availability strategies for critical components.

f. Troubleshooting and Incident Response:

* + Document common issues and their resolutions.
  + Include incident response procedures and escalation paths.

Remember that this is an overview of the design, and many details need to be fleshed out in practice. Also, the actual implementation may require adjustments depending on specific use cases and organizational requirements. The proposed documentation and code repositories would enable teams to deploy, manage, and monitor their products effectively while ensuring isolation and security in a multi-tenant Kubernetes platform.