**Solution to Scenario 2**

1. Recommendation as for the application environments (how many and why?) and how are they going to be separated

My recommendations for the application environment are as follows:

QA

DEV

PROD

STAGING

I

----- : Isolated Environment

: Build Artifact

1. **Development Environment (Dev):**

The development environment is the initial deployment stage following code build and preliminary testing. It serves as a sandbox for experimentation, allowing developers to verify that the application aligns with business requirements before advancing to the QA team.

1. **Quality Assurance Environment (QA):**

The QA environment is dedicated to conducting functional, regression, load, and security testing on the application. This phase ensures that the application meets business requirements, preserves existing feature functionality, and is free from security vulnerabilities that could impact users or the business.

1. **Staging Environment (Staging):**

The staging environment is a controlled setting designed to closely replicate the production environment. It is used for piloting changes, conducting smoke tests, performance evaluations, and user acceptance testing (UAT) on the application.

1. **Production Environment (Prod):**

The production environment is the live setting where end-users or clients access the application. It is the final deployment stage and must ensure optimal performance and security.

**Isolation:**

With regards to isolation, all four environments must be completely isolated from one another, with dedicated servers, network resources, and other infrastructure components. This segregation is critical to maintain integrity and prevent accidental data leaks between environments.

Example of the isolations are as follows:

1. **Network Segmentation:** Utilize distinct subnets or VLANs for each environment to enhance security and control.
2. **Access Controls:** Implement stringent access restrictions between environments.
3. **Dedicated Resources:** Ensure each environment possesses its own servers, databases, and storage solutions.

**d. Secret Management:** Maintain isolated secrets (e.g., API keys, passwords) for each environment to safeguard sensitive information. i.e separate vaults for each environment

1. Which tooling will you use for CI/CD and why? How to ensure no downtime during deployments?

For CI/CD, I would choose Azure DevOps due to its comprehensive and integrated toolset that supports the entire development lifecycle. It offers a wide suite of services including Azure Repos for version control, Azure Pipelines for CI/CD, Azure Boards for project management, Azure Test Plans for testing, and Azure Artifacts for package management. These services ensure a seamless workflow from code commit to deployment.

Additionally, Azure pipelines provide powerful CI/CD capabilities with support for YAML-based pipeline definitions as well as classical task definition. It also supports a wide range of programming languages build and deployment tasks, making it highly versatile. Whether you’re working with .NET, Java, Node.js, or Python, Azure DevOps can handle it. Additionally, it integrates well with other tools and services, both from Microsoft and third parties.

Azure DevOps can be used for both cloud-based and on-premises deployments. It also has support for deployments to Kubernetes clusters. It also boasts a vast and growing marketplace which holds a vast array of plugins and integration extending the capability of you DevOps Workflow.

For the application in question, I propose deployment to a Kubernetes platform as this has features, we can leverage to ensure zero downtime during deployment as well as improving reliability and scalability of the solution. On Kubernetes we can leverage a combination of Pod Disruption Budgets (PDBs) and Rolling Update Strategy.

Pod Disruption Budgets ensure that a minimum number of pods are always available during maintenance or updates. This prevents the application from becoming unavailable by maintaining a specified number of running pods while rolling update further enhance this by gradually replacing old pods with new ones. Instead of stopping the old pods immediately, new pods are brought up first. If the new pods are running correctly, traffic is then redirected to them before the old pods are terminated.

By utilizing both, we can achieve minimal downtime and maintain application availability during deployments.

1. What are the additional tooling you need to supplement the application with to ensure it runs smoothly on production? (e.g. from observability)

Additional tooling required:

1. **Service Mesh:** Installing service mesh like Istio or Linkerd helps to enhance observability within the deployed cluster by managing all communication between services and pods. It provides granular control and visibility into traffic routing, traffic splitting, encryption for all communication, distributed tracing and detailed telemetry data including metrics, logs and traces for all service-to-service communication. Both solutions offer dashboards to visualize the service topology, showing how requests flow through the system and highlighting areas of concern.
2. **Monitoring:** Installing monitoring tools like Datadog agent or EFK stack (Elasticsearch, Fluentd, Kibana) on the cluster provides comprehensive observability for both the platform and the applications running on it. Both solutions can collect metrics across the entire cluster, including CPU and memory usage, disk I/O, network traffic, and more. These metrics help in identifying resource bottlenecks and scaling needs.

Also, they support application performance monitoring (APM), collecting metrics such as request latency, error rates, and throughput. This helps in pinpointing performance issues within your applications.

They both support log aggregation and analysis making it easier to search, analyze, and visualize log data. You can also set up alerts based on specific metrics or log patterns to notify of potential issues.

1. **Container management Platform:** Utilizing container management platforms like Rancher and Lens provides a unified view and control over your Kubernetes clusters, enhancing observability and manageability at scale. Their key features involve providing a single pane glass to manage multiple Kubernetes clusters, providing a centralized dashboard to monitor and control cluster resources, workloads, and configurations. These platforms provide detailed visualizations of cluster health & resources, including nodes, namespaces, pods, and services. This helps in understanding resource allocation and utilization.
2. Networking & dns-records management and networking protection rules?

Utilizing **calico** on Kubernetes can enforce network protection rules through Kubernetes network policies which are applied at the pod level. These policies allow you to define rules governing ingress and egress traffic ensuring only authorized communication occurs between pods, namespaces, and external resources.

Calico extends these capabilities by supporting global network policies which provide cluster-wide control and fine-grained rules like CIDR-based filtering. Additionally, Calico’s support for BGP allows for seamless integration with external networks, providing enhanced routing capabilities and enabling direct communication between Kubernetes services and external system.

1. Mechanisms to ensure reliability and scalability?

To ensure reliability and scalability within a Kubernetes cluster several mechanisms are employed.

For reliability we can explore the below:

1. **Resource Limits and Requests:** It defines the minimum and maximum CPU and memory resources a pod can utilize thereby, helps to prevent resource contention and ensure fair distribution of cluster resources among workloads.
2. **Probes:**

* **Liveness Probes:** Check if an application is running.
* **Readiness Probes:** Determine if a pod is ready to handle traffic.
* **Startup Probes:** Ensure that an application has started successfully.

These probes are critical for maintaining the health and availability of applications.

1. **Pod Disruption Budgets (PDBs):** It helps tomaintain a minimum number of pods during voluntary disruptions, like node maintenance thereby, ensuring application availability and reliability during such events.

For scalability we can explore the below:

1. **Horizontal Pod Autoscaling (HPA):** Is a native Kubernetes resource that helps toAutomatically adjusts the number of pod replicas based on observed CPU utilization or custom metrics thereby enabling applications to scale out to handle increased load and scale in during low demand, optimizing resource usage.
2. **Integration with Third-Party Tools (e.g., KEDA):** This provides advanced scaling capabilities based on event sources beyond CPU and memory metrics. An example includes scaling based on messages in a queue or custom metrics from external systems like Prometheus. This enhances Kubernetes’ native scaling capabilities, allowing for more dynamic and responsive scaling strategies.

These collective mechanisms ensure that applications remain reliable and scalable, adapting to varying workloads while maintaining optimal performance and resource efficiency.

1. Alerting considerations?

Below are key considerations when setting up alerting for applications deployed on a cluster:

1. **Storage Utilization Alerts:**

* Set up alerts for persistent volume claims (PVCs) reaching 80% of their allocated storage capacity to provide early warnings on potential storage exhaustion.
* Configure alerts for failed attempts to provision new persistent volumes (PVs) or bind to PVCs which can indicate misconfigurations or underlying infrastructure issues.

1. **CPU and Memory Utilization:**

* Configure alerts for pods exceeding 85% of their CPU or memory requests for a prolonged period to identify pods that may require resource adjustments or optimization.
* Monitor and alert pods that consistently reach their resource limits, as this may suggest under-provisioning and the need for scaling out.

1. **Cluster-Wide Resource Contention:**

* Set up alerts for nodes experiencing resource pressure, such as high CPU, memory, or disk usage which can indicate the need for scaling the cluster or rebalancing resources.

1. **Log Analysis Alerts:**

* Configure alerts for specific critical error messages or patterns in logs, focusing on application-specific errors that require immediate attention.

1. **Network Policy Violations:**

* Set up alerts for any traffic that violates defined Calico network policies, indicating potential security breaches or misconfigurations.
* Monitor and alert unauthorized or suspicious attempts to access external networks or services from within the cluster as these may signal a compromised pod or malicious activity.