SCENARIO 1

1. Challenges in applying KMS key rotation

- **Multi-account and multi-service coordination challenges:** Each environment (dev, prod, etc.) and each service within that environment (S3, RDS, DynamoDB, etc.) has its own key. This means we have to manage the rotation of many keys without service interruption.
- **Impact of External (BYOK) Keys Generated in On-Premise HSM and Application Across Different Services:** The potential impact stems from the interconnections and dependencies between services, in addition to the fact that key segregation can affect different AWS services. Rotation requires generating new keys in the HSM and sending them securely to AWS KMS, which adds operational and security complexity.
- **Key Aliases:** AWS services reference aliases, not direct key IDs, but rotation may require updating policies or configurations depending on how it is implemented.
- **Minimize Disruptions:** Ensure that rotation does not cause downtime in encrypted services (e.g., RDS, S3, DynamoDB).
- **Regulatory Compliance:** Verify that the rotation process complies with the regulator's specific requirements and is auditable.

2. High-level steps to apply rotation

1. **Inventory and Planning:**
    - Identify all KMS keys (dev-s3, prod-rds, etc.) in the security account.
    - Define a maintenance window and rotation order (e.g., dev → int → prod).
2. **Generation of New Keys in HSM:**
    - Create new cryptographic material for each key in the on-premise HSM.
    - Package and protect the material for shipment to AWS KMS.
3. **Import to AWS KMS:**
    - Use ImportKeyMaterial in AWS KMS with the new external keys.
    - Assign the same aliases to the new keys (requires careful handling to avoid conflicts. This way, applications can continue to use the keys even after they have been rotated).

4. **Configuration Updates:**
    - For S3: Ensure that buckets continue to use the correct alias (usually automatic if an alias is used).
    - For RDS and DynamoDB: Verify that the KMS keys referenced in the encryption are updated if they use a direct ID (not an alias).
5. **Testing and Validation:**
    - Verify that new data is encrypted with the new key.
    - It must be confirmed that existing data remains accessible (non-destructive encryption).
    - Perform a planned rollback if problems arise.
6. **Secure Deletion of Old Keys:**
    - Once functionality is confirmed, schedule the deletion of the previous keys after the required grace period.

---

3. Monitoring resources without rotation applied

Use AWS Config with custom or managed rules:

- **AWS Config Rules:**
    - kms-key-not-rotated: Predefined rule to detect KMS keys not rotated within the defined period.
    - Create custom rules to validate that RDS, DynamoDB, and S3 use the correct key version.
- **AWS Security Hub:**
    - Integrate findings from AWS Config and enable security controls related to KMS (e.g., [KMS.1]).
- **Amazon EventBridge + AWS Lambda:**
    - Automate detection and notification when a resource uses an old key.
- **Key Tagging:**
    - Tag keys with a rotation date and use AWS Resource Groups to inventory resources by key.

4. Protecting key material during transport from HSM to KMS

The best practice is to follow the AWS KMS BYOK (Bring Your Own Key) process:

1. **Generate Key Package in HSM:**
    - Create a symmetric key in the HSM.
    - Use get-public-key from the KMS exchange key (obtained via GetParametersForImport).
2. **Encryption of Key Material:**
    - Encrypt the symmetric key with the KMS public key **inside the HSM (never export unencrypted material).**
3. **Secure Transport:**
    - Transmit the encrypted material via TLS/SSL connections (AWS KMS API/SDK).
    - Use AWS CloudHSM or an HSM connected via AWS Direct Connect/VPN to reduce internet exposure.
4. **Import Process:**
    - Call ImportKeyMaterial with the import token and the encrypted material.
    - AWS KMS decrypts it internally using its private key.
5. **Secure Deletion of Intermediate Material:**
    - Delete plain key material from the HSM and any traces in logs after import.

Additional Recommendation:
Consider using AWS CloudHSM integrated with KMS to simplify BYOK management and rotation, reducing the complexity of manual transport.

SCENARIO 2

1. Weaknesses in the current architecture

- All APIs are public by design, even internal ones, which unnecessarily increases the attack surface.
- Lack of segmentation between public and private APIs, exposing internal services to the Internet unnecessarily.

- Sole reliance on WAF/Shield as a protection layer, without defense in depth at the regional or service level.
- Risk of direct attack on regional API Gateway endpoints if an attacker discovers the regional URL, bypassing CloudFront and the global WAF.
- Potential performance degradation for internal calls, which must go out to the Internet and back in through CloudFront → API Gateway.
- Centralized authorization management in a single Lambda Authorizer, which can become a bottleneck or single point of failure.
- Lack of granular visibility and control between internal and external traffic.

2. New architecture to separate internal and mixed APIs

Proposal for a segmented architecture:

1. **Public/Mixed APIs** (external and/or internal use):
    - Maintain current exposure: api.allianz-trade.com → CloudFront → WAF → Regional API Gateway → Backend.
2. **Internal APIs** (internal use only):
    - Move to API Gateway VPC Endpoints (Private API).
    - Create private APIs in API Gateway with a VPC Endpoint (execute-api).
    - Expose them to the internal network via AWS PrivateLink.
    - Internal calls are made within the VPC, without going out to the Internet.
    - They can continue using the same internal domain (e.g., api-internal.allianz-trade.priv) with Route 53 Resolver.
3. **Optimization of internal traffic to mixed APIs:**
    - To prevent internal traffic from going over the Internet, configure Route 53 Resolver to redirect api.allianz-trade.com from the VPC to a closer CloudFront Regional Datacenter.
    - Another option: use API Gateway Private Integration for certain internal routes.

Resulting Architecture:
External → CloudFront → Public API Gateway → Backend
Internal → VPC Endpoint → Private API Gateway → Backend (Internal APIs)

Internal → Route53 Resolver → CloudFront (local edge) → Public API Gateway → Backend (Mixed APIs)

3. Configuring CloudFront for path-based routing

Use Behaviors in CloudFront:

1. Create CloudFront distributions that point to the regional API Gateway as the origin.
2. Define multiple behaviors with different path patterns:
   Path Pattern: /team1/* → Origin: api-gw-team1.execute-api.region.amazonaws.com
   Path Pattern: /team2/* → Origin: api-gw-team2.execute-api.region.amazonaws.com
   Path Pattern: /team3/* → Origin: api-gw-team3.execute-api.region.amazonaws.com
   Default (*) → Main origin
3. Configure the origin in CloudFront as a Custom Origin with:
   - Protocol: HTTPS
   - Origin Domain: API Gateway endpoint
   - Origin Path: (optional) for prefixes
   - Headers: Enable Host header forwarding so API Gateway validates correctly.
4. Use Lambda@Edge if route transformation or more complex routing logic is needed.

4. Protecting regional API Gateway endpoints

To prevent direct traffic that bypasses CloudFront/WAF:

1. **Use API Gateway Resource Policy** to restrict access only from:
   - The CloudFront distribution (CloudFront IP ranges).
   - Internal networks/VPC (for controlled direct access).

Example policy:

json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "execute-api:/*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": ["CloudFront IP ranges", "Internal IPs"]
        }
      }
    }
  ]
}
```

2. **Validation with custom headers:**
    - Configure CloudFront to add a secret header (e.g., X-Origin-Verify).
    - In API Gateway, create a Lambda Authorizer or Request Validation that rejects requests without that header.
3. **OAuth Scopes or API Keys:**
    - Require an API Key for direct access (if necessary for some clients).
    - CloudFront can send the API Key automatically.
4. **Additional regional WAF:**
    - Associate AWS WAF also at the regional API Gateway level for defense in depth.
5. **Custom Domains:**
    - Use only Custom Domain Names in API Gateway and do not expose the execute-api... domain.
    - Configure CloudFront as the sole public entry point.

Final Recommendation:
Implement AWS Network Firewall or Security Groups at the VPC level to filter outgoing/internal traffic and ensure that only authorized services can connect to API Gateway from the internal network.