Background:

A start-up company wants to host its Python and React-based application (Backend: Python API and Frontend React) using AWS. But they are not familiar with the AWS cloud platform. They want to ensure that the application is secure, scalable, highly available, and cost- efficient. As a solutions architect, you have to design a proper solution to meet their below requirements.

Goal:

To architect a solution that is secure, scalable, highly available, and cost-effective using AWS.

Requirements:

* They are concerned about the security of the application, so they have decided to isolate their network from the rest of the customers virtually. Set-up a secure virtual network where the only frontend of application is accessible by users and not the database.
* Execute the React application code using AWS Elastic Beanstalk. Ensure that the source code of Web application is automatically picked, pushed to the master branch, and deployed on the servers.
* Ensure all the UI images served to the frontend application code are provisioned via a secure storage unit.
* There should be enough backups for both the Web and Database server, so if the set-up crashes, we can launch a new one from the disaster recovery backups.
* They are uncertain about the traffic pattern that how low or high it can be, so they want the Web application to be running on at least two EC2 instances all time, and when there is a high load, they must burst up to four instances in total.
* The Web application should be highly available, even if any VM fails to respond to queries, there should be a mechanism to switch the connection to another healthy VM automatically.
* Automate the download all the activity logs into a CSV file, create a stream of data, analyze it, and display it via a dashboard.
* The Web application should also be cached globally, so users worldwide can access it with low latency.

To meet the requirements of a secure, scalable, highly available, and cost-effective solution using AWS, you can follow the architecture outlined below:

1. **Secure Virtual Network (VPC):**
   * Create a Virtual Private Cloud (VPC) to isolate your network from other customers. This provides a secure environment for your application.
   * Configure subnets within the VPC: public subnets for the frontend application and private subnets for the database.
   * Use security groups and Network Access Control Lists (NACLs) to control inbound and outbound traffic to the subnets.
2. **React Application with AWS Elastic Beanstalk:**
   * Set up a Git repository to host your React application code.
   * Configure a webhook or AWS CodePipeline to automatically trigger the deployment process whenever changes are pushed to the master branch.
   * Use AWS Elastic Beanstalk to deploy and manage your React application. Elastic Beanstalk handles infrastructure provisioning, deployment, and scaling for you.
3. **Secure Image Storage:**
   * Utilize AWS S3 (Simple Storage Service) to store and serve UI images securely.
   * Configure appropriate permissions and access control to ensure only authorized users can access the images.
   * Enable encryption at rest for the S3 bucket to enhance security.
4. **Backup and Disaster Recovery:**
   * Set up automated backups for both the web servers and the database.
   * For the web servers, you can use Amazon Machine Images (AMIs) to create snapshots, allowing you to launch new instances from these backups in case of failures.
   * For the database, use Amazon RDS (Relational Database Service) and enable automated backups and Multi-AZ deployment for high availability and automatic failover.
5. **Auto Scaling for Web Application:**
   * Set up an Auto Scaling Group (ASG) with a minimum of two EC2 instances to ensure high availability.
   * Configure scaling policies based on CPU utilization or other metrics to automatically adjust the number of instances.
   * Define a maximum limit of four instances to handle high traffic loads.
6. **Load Balancing and High Availability:**
   * Use an Elastic Load Balancer (ELB) to distribute incoming traffic across the EC2 instances.
   * Configure health checks to monitor the health of the instances. If an instance becomes unhealthy, the ELB will automatically route traffic to healthy instances.
   * Enable Elastic IP addresses or use Route 53 for DNS failover to switch traffic to another healthy instance if a VM fails to respond.
7. **Activity Logs and Dashboard:**
   * Enable AWS CloudTrail to capture activity logs across your AWS resources.
   * Set up an Amazon Kinesis Data Firehose to stream the CloudTrail logs to Amazon S3 or Amazon Redshift for storage.
   * Use a data analytics service like Amazon Athena or Amazon QuickSight to analyze the logs and create a dashboard to display the insights.
8. **Global Caching:**
   * Utilize Amazon CloudFront, a global Content Delivery Network (CDN), to cache and serve your web application content with low latency worldwide.
   * Configure CloudFront to distribute your static assets and provide edge caching for dynamic content.

By following this architecture, you can achieve a secure, scalable, highly available, and cost-effective solution using AWS services.

**Solution 2:**

To meet the requirements of a secure, scalable, highly available, and cost-effective solution using AWS for hosting a Python and React-based application, you can follow the architecture outlined below:

1. **Secure Virtual Network (VPC):**
   * Create a Virtual Private Cloud (VPC) to isolate your network from other customers. This provides a secure environment for your application.
   * Configure subnets within the VPC: public subnets for the frontend application and private subnets for the backend database.
   * Use security groups and Network Access Control Lists (NACLs) to control inbound and outbound traffic to the subnets.
2. **React Application with AWS Elastic Beanstalk:**
   * Set up a Git repository to host your React application code.
   * Configure a webhook or AWS CodePipeline to automatically trigger the deployment process whenever changes are pushed to the master branch.
   * Use AWS Elastic Beanstalk to deploy and manage your React application. Elastic Beanstalk handles infrastructure provisioning, deployment, and scaling for you.
3. **Secure Image Storage:**
   * Utilize AWS S3 (Simple Storage Service) to store and serve UI images securely.
   * Configure appropriate permissions and access control to ensure only authorized users can access the images.
   * Enable encryption at rest for the S3 bucket to enhance security.
4. **Backup and Disaster Recovery:**
   * Set up automated backups for both the web servers and the database.
   * For the web servers, you can use Amazon Machine Images (AMIs) to create snapshots, allowing you to launch new instances from these backups in case of failures.
   * For the database, use Amazon RDS (Relational Database Service) and enable automated backups and Multi-AZ deployment for high availability and automatic failover.
5. **Auto Scaling for Web Application:**
   * Set up an Auto Scaling Group (ASG) with a minimum of two EC2 instances to ensure high availability.
   * Configure scaling policies based on CPU utilization or other metrics to automatically adjust the number of instances.
   * Define a maximum limit of four instances to handle high traffic loads.
6. **Load Balancing and High Availability:**
   * Use an Elastic Load Balancer (ELB) to distribute incoming traffic across the EC2 instances.
   * Configure health checks to monitor the health of the instances. If an instance becomes unhealthy, the ELB will automatically route traffic to healthy instances.
   * Enable Elastic IP addresses or use Route 53 for DNS failover to switch traffic to another healthy instance if a VM fails to respond.
7. **Activity Logs and Dashboard:**
   * Enable AWS CloudTrail to capture activity logs across your AWS resources.
   * Set up an Amazon Kinesis Data Firehose to stream the CloudTrail logs to Amazon S3 or Amazon Redshift for storage.
   * Use a data analytics service like Amazon Athena or Amazon QuickSight to analyze the logs and create a dashboard to display the insights.
8. **Global Caching:**
   * Utilize Amazon CloudFront, a global Content Delivery Network (CDN), to cache and serve your web application content with low latency worldwide.
   * Configure CloudFront to distribute your static assets and provide edge caching for dynamic content.

By following this architecture, you can achieve a secure, scalable, highly available, and cost-effective solution using AWS services.