Concept integration

Benefits of caching

Retrieving data, from a database or an S3 bucket for example, can be costly if it's done for lots of users constantly. One way to reduce the need for data retrieval is to implement caching. Caching in computing means temporarily storing frequently accessed data in a more accessible location. There are two main benefits that caching offers for our application: speed and cost.

Cache is a high-speed data storage layer. Reading from a cache is typically much faster than querying a relational database, which our application uses. Cache also is physically located closer to the user than the backend database. These properties of caching makes retrieving data much faster. The speed allows us to provide users a more responsive experience.

Some cost savings can be gained by using caching. AWS **CloudFront** is a more efficient way of retrieving data than going all the way to the EC2 instance and retrieving what is needed. Some in-memory caching types however may be more expensive than simply reading from the database. However, since caching provides other features as well, such as session management, that we would need to implement anyway, the benefits may outweigh the increased cost.

Challenges related to caching

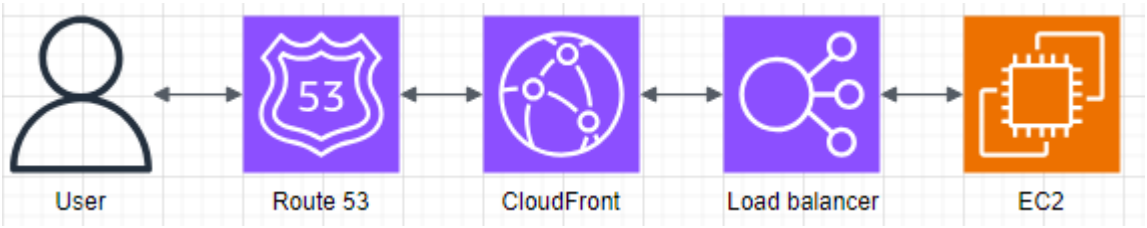
Stale data is a drawback of caching. It means that the information presented to the user, might have changed after it was cached. In our application, this could mean that if an event organizer changes the time for an event, cached data displayed to the user might not be correct data.

Another challenge in caching is whether it is useful in an application that relies heavily on search results. Our application allows users to browse and search events and organizers. Since the data expected user to event-organizer ratio is expected to be heavily slanted towards users, we expect caching to still provide benefits. This is because the same events and organizations will be loaded repeatedly.

Implementating caching in Volunteer management

Our implementation runs the website in EC2 instances. This lets us make use of the Content delivery network, CDN. With AWS CloudFront, content like images and web components can be stored at an edge location. With CloudFront, content can be retrieved from an edge location instead of the origin server, which might exist much further away geographically.

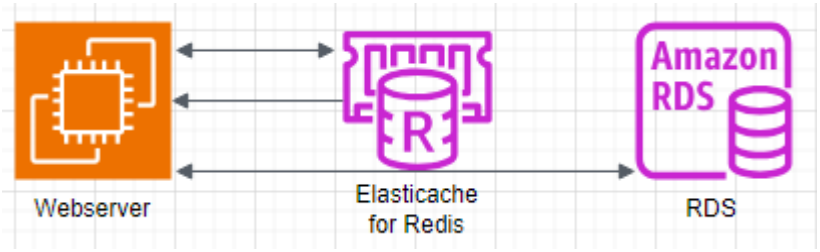
CloudFront is greatly customizable. For CloudFront to operate, a distribution must be configured. The distribution tells CloudFront the origin of the content to be cached, which content is cached and how long content will be cached, among other things.



1Caching with CloudFront shortens the distance from user to data

Another caching strategy that needs to be implemented is storing user session information. Storing user sessions reduce the need to validate logins repeatedly. This offers a much-improved user experience as well as reduced workload for validation.

Since our application uses a relational database, **Amazon ElastiCache for Redis** can be used to store session information. Redis is an in-memory data store with a high performance. Elasticache supports lazy loading, which is perfect for a social media application. A Redis cluster can be launched using the ElastiCache console on AWS.



2Lazy loading only queries the database after a cache miss

Downside of Redis is that it requires lots of RAM to function. This can lead to high costs. Alternative to Redis is using sticky sessions. Sticky sessions is a feature of the **Elastic LoadBalancing Load Balancer**. ELB can direct traffic to the instance that is managing a user's session. This can lead to lost sessions when instances are terminated.

Benefits of CI/CD practices

There are several actions that can be taken to improve CI/CD practices that are supported by AWS.

CloudFormation is a service that can be used to create our AWS architecture as code. Having a deployable code representation of our architecture allows some key benefits. One of the benefits is that the IaC-code can be saved externally on GitHub, for example.

Another benefit is that deployments can be automated. Automated deployment of AWS infrastructure helps reduce errors caused by manually configuring the environments. Configuring multiple versions of large environments for testing, production, and other purposes, is sure to produce variations between them. Option to roll back faulty deployments makes using CloudFront relatively safe and adds to the appeal.

One more benefit of CloudFormation is that it provides a visual representation of our architecture. This tool can be used to observe illogical structures in the architecture. In addition to visually observing the bigger picture, code can be inspected for more specific details.

Several tools exist to automate patching operating systems, EC2 configurations etc. It would not be feasible to update hundreds of EC2 instances by connecting to each instance using PuTTY and running commands by hand.

Another way to increase automation is to automate software deployments to EC2 instances. In our project, webservers are run on EC2 instances. Manually building and deploying software in AWS can be a time-consuming task. A pipeline to deploy code committed to Git directly on to EC2 can dramatically improve developer experience.

Challenges related to CI/CD

A big challenge for our team is our lack of experience in DevOps engineering. Learning about all the different pipelines, automation strategies and how they integrate with version control systems and AWS can easily cause information overflow. There are tons of completely different CI/CD approaches. What we must do is select few of the best ways that help us in developing the application.

One challenge with IaC in AWS is that not all services are fully supported by CloudFormation. Some new features added to services may not be available for CloudFormation. This could create issues where the template does not fully represent the architecture.

Implementing CI/CD practices

To make full use of CloudFormation and its many benefits, we need a code representation of our infrastructure. There are 2 quick ways to get started with using CloudFormation. One way is to use ready templates created by AWS or 3rd party vendors. These can provide a starting point for architecture that can be modified to fit our needs. The second option is to use CloudFormer tool to create the IaC-code from our existing environment. CloudFormer is no longer supported by AWS. Popular 3rd party tools exist to replace CloudFormer, such as Former 2, an open-source tool that scans your AWS environment and generates IaC-code from it.

Tools may be helpful to learn by doing, but our choice is to use the designer provided in CloudFormation. Yaml editor is also helpful as we become experts on the subject. Any architecture code and change sets will be saved in a GitHub repository for version control and storing. This yaml is then used to create stacks and deploy our environment.

Deploying both CloudFormation yaml and webserver happens through pipelines that link our GitHub and AWS resources. GitHub integration is possible with the AWS CodeDeploy service. A secure connection will be established to allow GitHub to trigger actions in the AWS environment.

To establish a connection between GitHub and AWS, IAM role can be used. GitHub uses secrets to handle integrations. By specifying the Amazon resource of the IAM role, the process can be defined to have all the rights it needs to complete the task. In addition to the IAM role, during the stack creation a GitHub repository name must be specified. This can be achieved by using custom parameters in the CloudFormation template.

Build instructions for the webserver code and a specific S3 bucket are specified in Github. The S3 bucket will be used to deliver deployment information to AWS CodeDeploy. Furthermore, the yml needed to deploy on EC2 instances should be in the repository. This can include scripts that need to be run on various stages of the deployment.

After manually verifying the connectivity works as intended, the process can be made fully automatic. In a fully automated pipeline, code committed to Git is immediately propagated towards AWS.