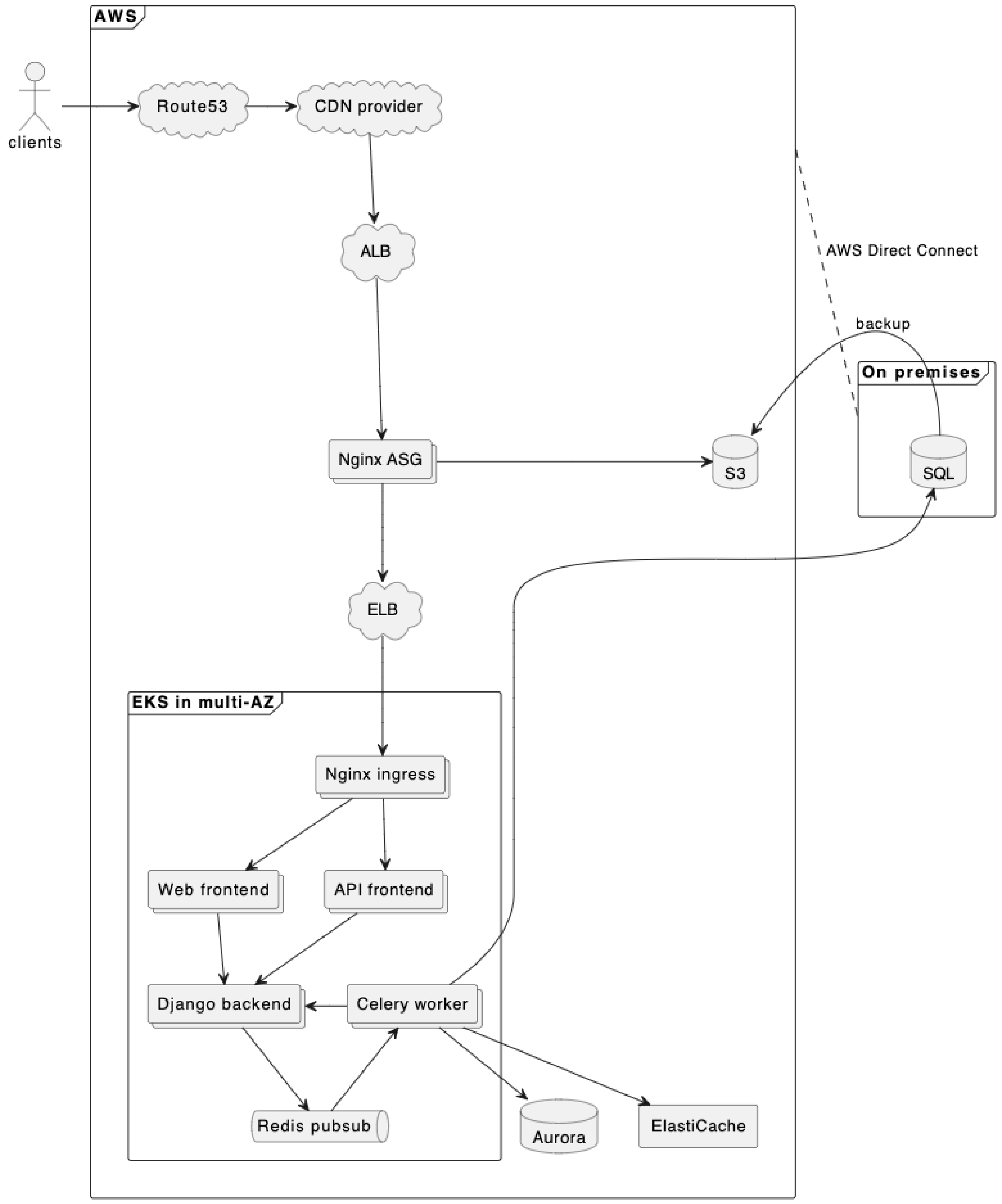

Notes on components and services:

- Cloud Provider is AWS, Tech stack uses Python Django framework as backend, the frontend solution could use one of React, Angular or VueJS

- Route53 provides DNS service to serve different DNS for various client categories, CDN provider is whether CloudFront in AWS or Akamai by 3rd part to cache data, both of them set origin to ALB
- Nginx ASG behinds ALB serves both static and dynamic contents routing, route to S3 for static contents and next ELB exported by Nginx ingress controller for dynamic requests. It also rewrites requests to appropriate paths for Nginx ingress reverse proxy
- Nginx ingress provides reversing proxy, two main paths or endpoints are Web frontend which serves requests from website/mobile app and API frontend which serves api requests from e.g. ticket board
- The requests go to Django backend, which will publish a task in Redis PUBSUB queue, and Celery worker fetches a task to run, the task will use API to read data from the database. To optimize the performance, API can be designed as reading cache from ElastiCache firstly, if the cache is not hit, then it will directly read data from Aurora in AWS and SQL server on-premises, once it gets data back then writes data to ElasitCache, that is called lazy loading mechanism
- There is AWS Direct Connect between AWS and On-premise network so that services in AWS could access SQL server on-premise. Moreover, the regular backup of SQL server data is stored in AWS S3 for recovery purposes.
- EKS holds web services except Nginx. The services are stateless and defined with HPA so that they can scale-out and in based on load e.g. cpu, mem, or queue status

Release and Deployment:

- Services in EKS are running in CICD way, CI to check code lint, run UT, FT, integrated testing by using GitHub/Gitlab/Jenkins CI workflow, this is automatic; While CD is manual approval workflow in GitHub/Gitlab/Jenkins to deploy container based service in EKS. To achieve green/blue deployment, service mesh could be used to direct traffic based on needs or we could set up another EKS in another region, and manage traffic in CDN
- Nginx ASG deployment and configuration are managed by IaC tool like Ansible

Security:

- All data on rest or way shall be encrypted in following ways:
  - Storage of Database is encrypted and access point of Database is TLS
  - Instances' storage of Nginx ASG are encrypted. In case EKS uses self-managed nodes, the nodes' storage shall be also encrypted
  - Data in S3 side are encrypted and upload/download actions shall be based on secure connection, and S3 bucket public access shall be disabled
  - All credentials used by EKS services are stored in AWS SSM
  - Certificates are managed by AWS or CDN provider in case 3rd part CDN provider is chosen, the certificates are attached in ALB, Nginx ingress so that connections are encrypted
- Services' container images are stored in trusted provider e.g. AWS ECR so that vulnerable scanning can be done
- WAF and DDoS protection shall be enabled in CDN side

Performance:

- Nginx ASG side configures rate limit according to published SLO, proper response shall be sent to client when rate limit is reached
- Nginx ASG is auto scaled-out and in based on its CPU load or self-defined number of requests
- Django backend task handling mechanism is asynchronous by using pub-sub mode, the celery worker replicas can be scaled-out and in based on Redis queue status
- Both frontend and backend Pods are auto scaled-out and in based on their CPU or memory load
- Aurora read replicas are created to off-load read loads
- ElastiCache is used to cache data on both Aurora and on-premises SQL server, moreover cronjob can be created to cache data in ElastiCache from on-premises SQL server to improve hitting rates as reading data from on-premises side is time-consuming

Monitoring, alerting and traceability:

- The EKS services can be monitored by cloud native solution e.g. Prometheus stack
- INginx ASG, Database Aurora and ElastiCache could be monitored by AWS CloudWatch
- SQL server on-premise could be monitored by other tool e.g. Zabbix
- E2E of services monitoring can be done by e.g. Pingdom
- All of those monitoring tools can connected to Pageduty, which could be triggered by alert and do unified notification way
- As of traceability, all logs can be directed to AWS CloudWatch log groups so that engineers can take a detailed investigation
- AWS X-Ray or other cloud native solution can be used to profile services' capabilities so that engineers could easily detect the performance bottlenecks

Disaster Recovery:

- If cost is not concerned so much, additional EKS cluster can be set up in another region so that traffic can be switched to standby region in case of disaster in master region
- All cloud resources deployment and configuration shall be managed by IaC tool e.g. CloudFormation/Terraform and Ansible, so that the resources and services could be deployed quickly in another region in case of disaster in the master region to satisfy RTO. Of course, data in database shall be backup regularly to high available and durable storage e.g. S3, then the data could restored quickly to satisfy RPO

Trade-offs, everything has trade-off, following are only some examples:

- The Aurora read replicas improve read performance while bringing data consistency issue as read replicas use async-replication way with master
- Container in EKS solution benefits operation and scalability while it introduces more network layer to reduce network performance
- Content caching improves loading performance while brings possibility of reading unclean data; application caching improves reading performance from database in most cases while it brings more overhead if the cache is not hit