Task 1) Web app deployment and scaling

1.1 - Would use Docker and Docker-compose to dockerize(put in a container and run as an image) so that CI/CD can be painless. Eventually if we would had more than one container, something like modern microservices approach, we would use Docker-compose.

In that case it would necessary to install Docker so that it can be managed via CLI.

Create one folder named "docker" and in it two subfolders(app/web) where in both subfolders I would put dockerFile while in "web" one there would be dockerfile+nginx.conf, of course both dockerfile and nginx.conf would be populated with required configurations.

With docker-compose.yml in root of our app, we would dockerize our containers, update our database .yml file, and run docker-compose build>docker-compose up -d (eventually verify with "ps" switch)

Finally deploy Docker image with Amazon ECS to some Amazon instance type like EC2, and eventually use storage service like S3 for security, archive, backup or other purposes.

1.2 - In these circumstances we can use Amazon load-balancing services such as Amazon CloudWatch or create and configure "Scaling Policies" for our instances to manage thresholds.

Overall, we need load-based auto scaling so that traffic is observed and proper resources are applied for load in particular circumstances.

1.3 - We can implement staging or multi-staging environment for which there should be additional hosting space, domain>urls, ssl cert, db configurations, alternate SMTP and other necessary components for this purpose. Also, it can be automated during pipelining continuous integration/continuous deployment with various scripts and/or services like Jenkins, Circle or Travis.

By adding dependencies of testing frameworks such as Rake, RSpec, Capistrano or Cucumber to our project so we can make unit, integration, smoke and functional tests. Bundler can be used to manage Ruby Gems for our app.

Maybe helpful would be to use mock server such as Mirage who acts as a proxy for HTTP request and as such is addition to testing in general.

1.4 - Use OAuth2 protocol/server with OpenID Connect, and possibly use KeyCloak, furthermore in a proper way implement ID Token (JWT) or Access Token (Bearer token) to identify and authorize.

Eventually limit number of requests (per connection, user, application or context)

Not sure about "API First approach" to do it through Swagger and is that a common practice, or even use OpenAPI3 tools.

Task 2) Event aggregation and data warehouse

2.1 - Use tool like Beats to collect and FileBeat/Logstash (or similar) to parse data.

Maybe using regular expressions would aid the process.

2.2 - With tools like Logstash with pipeline that takes Apache logs as input, we can parse data to Elasticsearch to be able to aggregate.

Structure semantic logging may be of use but I don't understand it well or is it really useful.

2.3 - Couchbase, PostgreSQL or MongoDB databases would be useful for storing data with something like Elasticsearch.

Possibility of using ELK stack to monitor the whole process of collecting, parsing, transforming and visualizing data. Custom console based on Zepkin and ELK stack would be an ideal solution for complete metrics, analytics, monitoring and alerting.