Key Components

Cloud Provider
AWS

Infrastructure as Code
Provisioning - Terraform, Kops
AMI Package Management - Packer
Continuos Deployment - Ansible
Continuos Integration - Jenkins (Pipeline as Code)

Containerisation
Docker
Kubernetes and its related tools like helm charts

Registry
DockerHub
Chart Museum

Source Code
GitHub

Collaboration/Notification
Slack, PagerDuty,SNS, emails etc.,

Datastore
AWS Aurora PostgreSQLDB

Vulnerability Scanning tools
Docker and Kubernetes Scanner

Monitoring/Observation
Grafana, Prometheus, Prom Alert Manager, EFK Stack

Explanation

- We can choose any cloud provider you want to, I have decided with AWS as they are more stable to rely on and we can build the entire stack without much hassles.
- I have chose to use maximum of CNCF Cloud Native Projects wherever possible, to **avoid costs** and major developments have already been taken place and also the open source contributions are enormous in DevOps world.
- All the key components that I have chosen are the projects which is contributed and managed regularly and they are **Cloud agnostic** in nature and they can **run on any cloud** for flexibility.
- Yet, If needed we can use the best paid products, may be for better monitoring and security of infra as these cannot be compromised at any cost.
- The application can easily scale, and it will be highly resilient and performs best HA capability using the architecture stack that I have provided.

How we deploy as a Code

STEP1 - INFRASTRUCTURE BUILDING

- AMI Package - Packer scripts for backing AMI Images can be maintained as code in Git and it is pretty straightforward, Lets initially start creating the basic AMI or backbone AMI image for running our chat apps or infra. We can update and create new AMIS using packer as we progress further, this is all done using code. The packer scripts can be written on JSON and YAML.
- Provisioning - terraform codes which we can write in HCL and save as source in Git. This will provision the entire AWS infrastructure that includes, EC2, ALB, R53, Datastores etc., We can either use direct terraform or we can use AWS SDKs and write our own code using python, node js etc.,
- Config Mgmt - Write Ansible roles, playbook, scripts etc., everything in ansible preferred YAML language for maintaining the infrastructure, and if there is something needs to added we can quickly add as code via sensible off course by reviewing it via PR process. Ansible makes sure that whatever is present on the source is applied on infra. It uses idempotent.
- Continuos Integration - Make the Jenkins DSL Code pipeline ready so that “developers” can use it themselves to deploy the Application code, in this case chat application. We can write Jenkins DSL and maintain it as code and give privilege to developers to run it as a single click.

STEP2 - Container Tech Stack

- Docker Images and DockerFiles and if needed docker-compose.yaml files should be written based on application and this can be maintained in code in Git and create the Docker images on Docker hub using CI tools like Jenkins and Circle CI
- Building Kubernetes clusters are easily done again coding and maintaining in git using Terraform, Kops etc., They are all JSON and YAML and can be easily maintained as source of truth. We can encrypt secrets in git using Sealed secrets which is special feature in K8s.
- Then we start by doing Creating the Helm Charts (Usually written on yaml and go programming language) can be maintained in code and they are written in line with application, in this case it is Chat app and also should match the requirements on K8s clusters to deployed.
- Publishing the charts and images into appropriate repos gives us a clear picture on what is being rolled out or what image is being built recently based on tags. We can use PostMan based API scripts and maintain in code to capture real time data.
- Docker and kubernetes scanning can be done with various open source tools that are available which will can quickly check vulnerabilities and report to us. We can write scripts and and maintain the same as source of truth.

STEP3 - Datastore

- I have chosen to use the AWS Aurora PostgreSQL as a database for building the chat application.
- This will be maintained via code. Terraform has provisions to build the entire datastore via code.
- Automatic snapshots and manual snapshots of Aurora RDS can be maintained using AWS Lambda python codes that will run regularly based on defined timelines.

STEP4 - Monitoring, Tracing , Logging and Observation

- I have chosen to helm charts which are again written in code and can easily fine tuned based on our requirements.
- ELK, EFK , Grafana, Prometheus, Jaegar and almost everything we can maintain using codes and we can use as per our requirements.

STEP5 - Development

- Make a way for developers to start developing on AWS Systems, Docker based systems, Kubernetes based systems.
  Example, create a framework in code in this case, create the chat application and let the developers develop it locally, so that they can regularly test on local laptops using `docker-compose` if docker engine or `minikube` if kubernetes.
  They are again code and maintained in source code so everyone is aware what is happening.