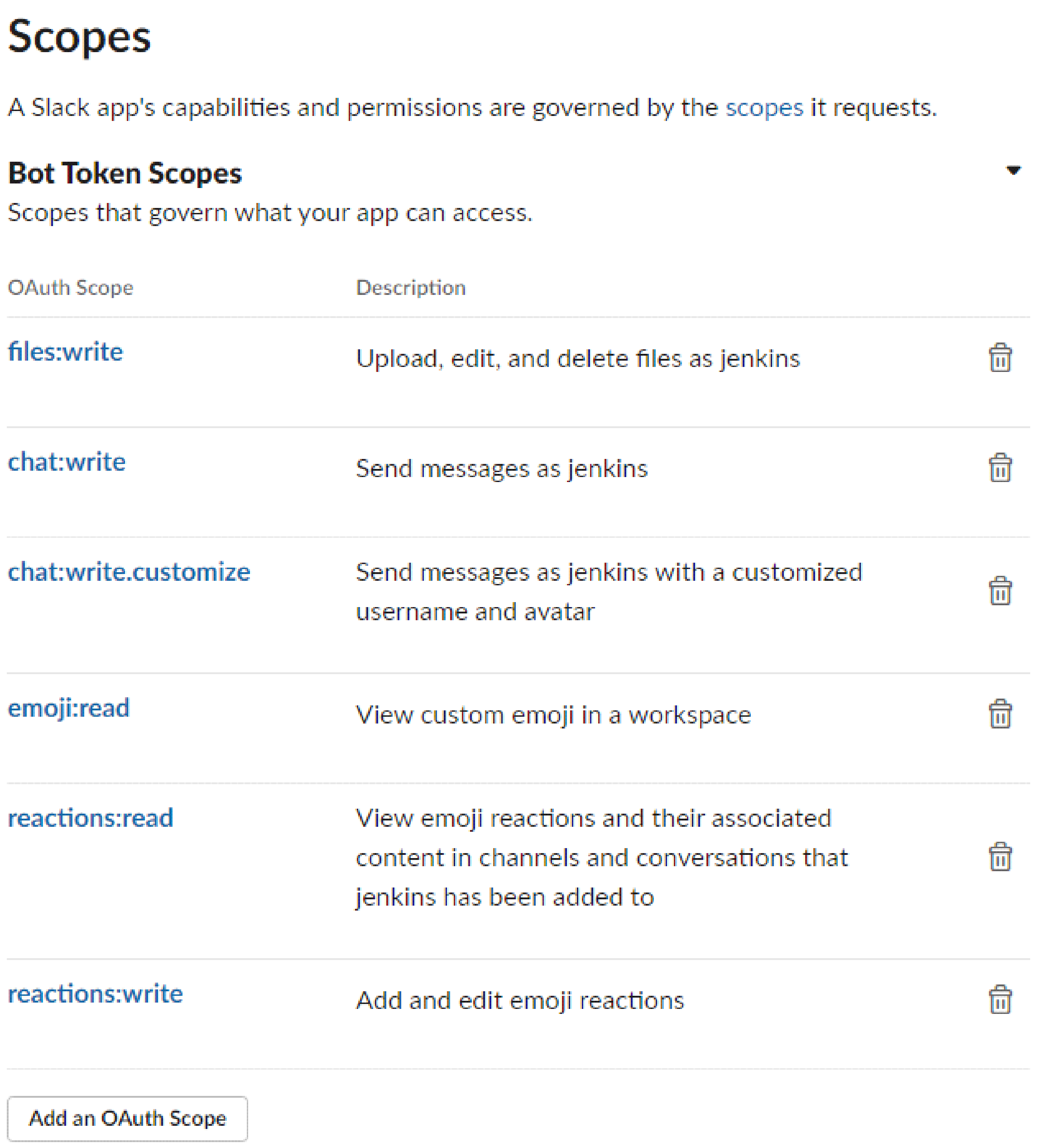
Custom jenkins alert

Create jenkins bot

Go to https://api.slack.com/apps and select

- Go to https://api.slack.com/apps and click "Create New App".
- Pick an app name, i.e. "Jenkins" and a workspace that you'll be installing it to.
- Click "Create App". This will leave you on the "Basic Information" screen for your new app.
- Scroll down to "Display Information" and fill it out. You can get the Jenkins logo from: https://jenkins.io/artwork/.
- Scroll back up to "Add features and functionality".
- Click "Permissions" to navigate to the "OAuth & Permissions" page.
- Scroll down to "Scopes". Under "Bot Token Scopes"
    1. Add chat:write Scope.
    2. (optional) Add files:write Scope if you will be uploading files.
    3. (optional) Add chat:write.customize Scope if you will be sending messages with a custom username and/or avatar.
    4. (optional) Add reactions:write Scope if you will be adding reactions.
    5. (optional) Add users:read and users:read.email Scope if you will be looking users up by email.
- (optional) Click "App Home" in the sidebar
    1. (optional) Edit the slack display name for the bot.
    2. Return to the "OAuth & Permissions" page.



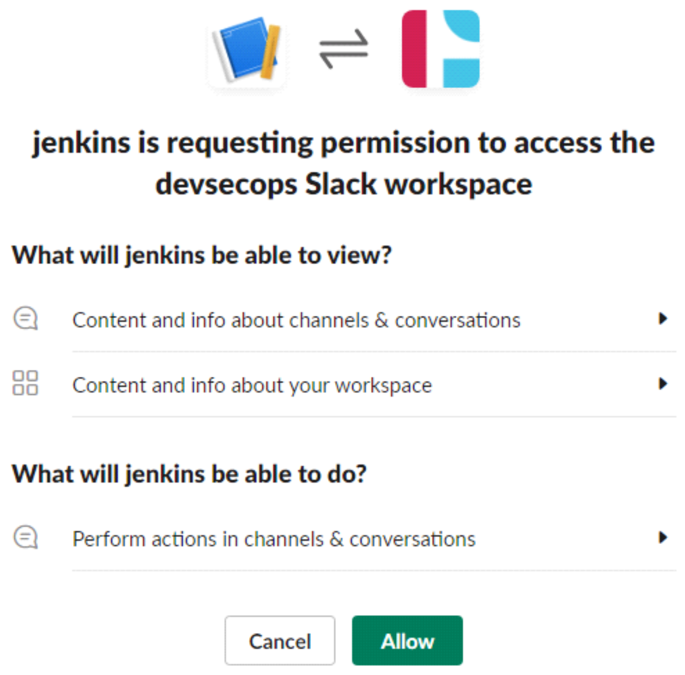
Scopes

A Slack app's capabilities and permissions are governed by the scopes it requests.

Bot Token Scopes

Scopes that govern what your app can access.

OAuth Scope	Description
files:write	Upload, edit, and delete files as jenkins
chat:write	Send messages as jenkins
chat:write.customize	Send messages as jenkins with a customized username and avatar
emoji:read	View custom emoji in a workspace
reactions:read	View emoji reactions and their associated content in channels and conversations that jenkins has been added to
reactions:write	Add and edit emoji reactions


jenkins is requesting permission to access the devsecops Slack workspace

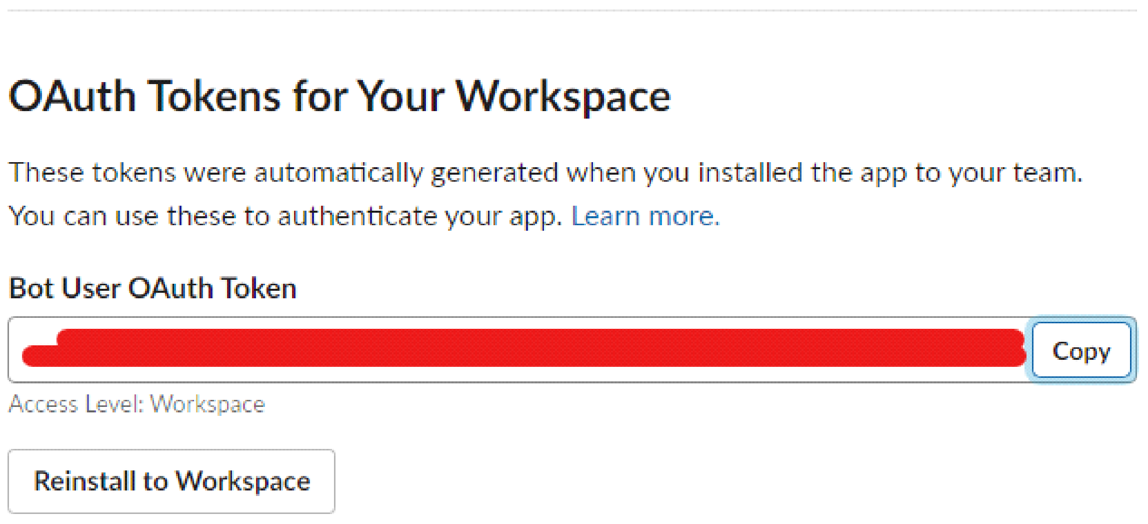
What will jenkins be able to view?

Content and info about channels & conversations

Content and info about your workspace

What will jenkins be able to do?

Perform actions in channels & conversations


OAuth Tokens for Your Workspace

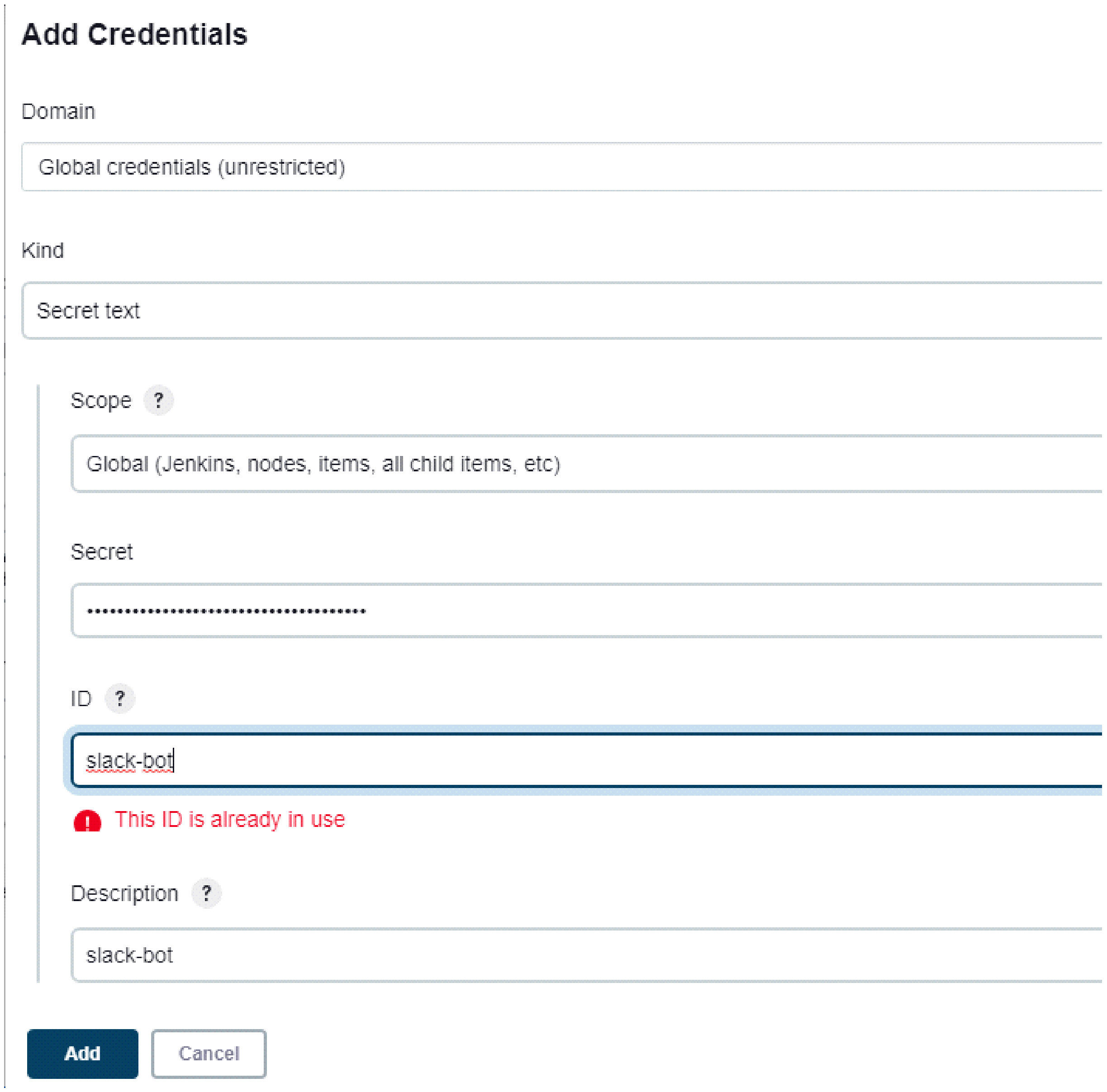
These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. Learn more.

Bot User OAuth Token

[redacted]

Access Level: Workspace

- At the top of the page, click "Install App to Workspace". This will generate a "Bot User OAuth Access Token".
- Copy the "Bot User OAuth Access Token".

- 

Add Credentials

Domain

Global credentials (unrestricted)

Kind

Secret text

Scope

Global (Jenkins, nodes, items, all child items, etc)

Secret

••••••••••••••••••••••••••••••••••••••

ID

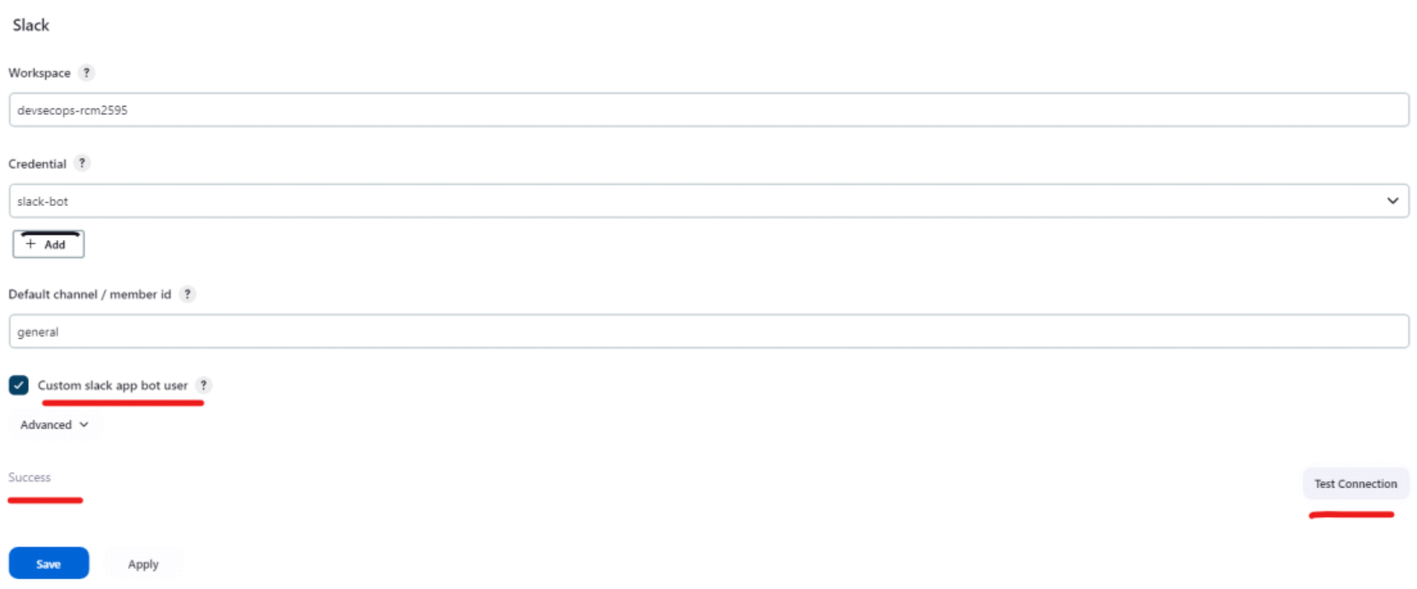
slack-bot

This ID is already in use

Description

slack-bot

- *On Jenkins*: Find the Slack configuration in "Manage Jenkins → Configure System".
    1. *On Jenkins*: Click "Add" to create a new "Secret text" Credential with that token.
    2. *On Jenkins*: Select the new "Secret text" in the dropdown.
    3. *On Jenkins*: Make note of the "Default channel / member id".
    4. *On Jenkins*: Tick the "Custom slack app bot user" option.
- Invite the Jenkins bot user into the Slack channel(s) you wish to be notified in.


Slack

Workspace ?

devsecops-rcm2595

Credential ?

slack-bot

+ Add

Default channel / member id ?

general

Custom slack app bot user ?

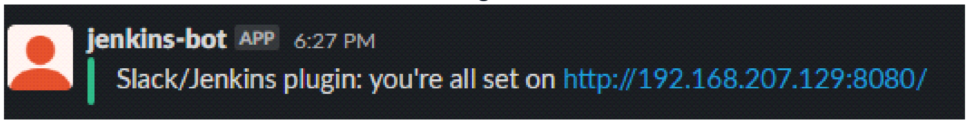
Advanced

Success

Test Connection

Save Apply

- *On Jenkins*: Click test connection. A message will be sent to the default channel / default member.


jenkins-bot APP 6:27 PM

Slack/Jenkins plugin: you're all set on http://192.168.207.129:8080/

We will receive this notification if test connection sucess.

Read more: https://plugins.jenkins.io/slack/#plugin-content-creating-your-app

Custom alert with Block Kit Builder

Block Kit is a framework for building Slack messages that allows developers to create modular and flexible message layouts. Block Kit Builder provides a visual interface for creating and customizing Block Kit messages, making it easier for non-developers to create messages without writing any code.

https://app.slack.com/block-kit-builder

Update vars/sendNotification.groovy

```
def call(String buildStatus = 'STARTED') {
 buildStatus = buildStatus ?: 'SUCCESS'

 def color

 if (buildStatus == 'SUCCESS') {
  color = '#47ec05'
  emoji = ':ww:'
 } else if (buildStatus == 'UNSTABLE') {
  color = '#d5ee0d'
  emoji = ':deadpool:'
 } else {
  color = '#ec2805'
  emoji = ':hulk:'
 }
```

```
// def msg = "${buildStatus}: `${env.JOB_NAME}` #${env.BUILD_NUMBER}:\n${env.BUILD_URL}"

// slackSend(color: color, message: msg)

 attachments = [
   [
     "color": color,
     "blocks": [
       [
         "type": "header",
         "text": [
           "type": "plain_text",
           "text": "K8S Deployment - ${deploymentName} Pipeline  ${env.emoji}",
           "emoji": true
         ]
       ],
       [
         "type": "section",
         "fields": [
           [
             "type": "mrkdwn",
             "text": "*Job Name:*\n${env.JOB_NAME}"
           ],
           [
             "type": "mrkdwn",
             "text": "*Build Number:*\n${env.BUILD_NUMBER}"
           ]
         ],
         "accessory": [
           "type": "image",
           "image_url": " https://raw.githubusercontent.com/sidd-harth/kubernetes-devops-security/main/slack-
emojis/jenkins.png",
           "alt_text": "Slack Icon"
         ]
       ],
       [
         "type": "section",
         "text": [
             "type": "mrkdwn",
             "text": "*Failed Stage Name: * `${env.failedStage}`"
           ],
         "accessory": [
           "type": "button",
           "text": [
             "type": "plain_text",
             "text": "Jenkins Build URL",
             "emoji": true
           ],
           "value": "click_me_123",
           "url": "${env.BUILD_URL}",
           "action_id": "button-action"
         ]
       ],
    [
      "type": "divider"
    ],
    [
      "type": "section",
      "fields": [
        [
          "type": "mrkdwn",
          "text": "*Kubernetes Deployment Name:*\n${deploymentName}"
        ],
        [
          "type": "mrkdwn",
          "text": "*Node Port*\n32564"
        ]
      ],
      "accessory": [
        "type": "image",
        "image_url": " https://raw.githubusercontent.com/sidd-harth/kubernetes-devops-security/main/slack-
emojis/k8s.png",
        "alt_text": "Kubernetes Icon"
      ],
    ],

    [
      "type": "section",
      "text": [
          "type": "mrkdwn",
          "text": "*Kubernetes Node: * `controlplane`"
        ],
      "accessory": [
        "type": "button",
        "text": [
          "type": "plain_text",
          "text": "Application URL",
          "emoji": true
        ],
        "value": "click_me_123",
        "url": "${applicationURL}:32564",
        "action_id": "button-action"
      ]
    ],
    [
      "type": "divider"
    ],
    [
      "type": "section",
      "fields": [
        [
          "type": "mrkdwn",
          "text": "*Git Commit:*\n${GIT_COMMIT}"
        ],
        [
          "type": "mrkdwn",
          "text": "*GIT Previous Success Commit:*\n${GIT_PREVIOUS_SUCCESSFUL_COMMIT}"
          ]
        ],
        "accessory": [
          "type": "image",
          "image_url": " https://raw.githubusercontent.com/sidd-harth/kubernetes-devops-security/main/slack-
emojis/github.png",
          "alt_text": "Github Icon"
        ]
      ],
      [
        "type": "section",
        "text": [
            "type": "mrkdwn",
            "text": "*Git Branch: * `${GIT_BRANCH}`"
          ],
        "accessory": [
          "type": "button",
          "text": [
            "type": "plain_text",
            "text": "Github Repo URL",
            "emoji": true
          ],
          "value": "click_me_123",
          "url": "${env.GIT_URL}",
          "action_id": "button-action"
        ]
      ]
    ]
  ]
 ]

 slackSend(iconEmoji: emoji, attachments: attachments)

}
```

This is a Groovy script designed to send a notification to a Slack channel based on the outcome of a Jenkins build. The alert uses the Jenkins Slack plugin to post messages to the Slack channel.
The script defines a function called call that takes an optional buildStatus parameter, which defaults to 'STARTED'. The function first sets buildStatus to 'SUCCESS' if it is not already defined.
Next, the function sets the color and emoji for the Slack message based on the buildStatus value.

If buildStatus is 'SUCCESS', the color is set to green ('#47ec05') and the emoji is set to ':ww:'.
If buildStatus is 'UNSTABLE', the color is set to yellow ('#d5ee0d') and the emoji is set to ':deadpool:'. Otherwise, the color is set to red ('#ec2805') and the emoji is set to ':hulk:'.

The Slack message is defined using an attachments array. The attachments array contains a single attachment object that specifies the color and content of the Slack message. The attachment contains several sections that provide information about the build, including the job name, build number, failed stage name, Kubernetes deployment name, node port, application URL, Git commit, Git branch, and GitHub repo URL.

The message is sent using the slackSend function with the iconEmoji and attachments parameters. The iconEmoji parameter specifies the emoji to use as the icon for the Slack message. The attachments parameter specifies the attachments array containing the message content.

Jenkinsfile - Add Slack Notification in Post Success

```
@Library('slack') _

/////// ****************************** Code for fectching Failed Stage Name ******************************
///////
import io.jenkins.blueocean.rest.impl.pipeline.PipelineNodeGraphVisitor
import io.jenkins.blueocean.rest.impl.pipeline.FlowNodeWrapper
import org.jenkinsci.plugins.workflow.support.steps.build.RunWrapper
import org.jenkinsci.plugins.workflow.actions.ErrorAction

// Get information about all stages, including the failure cases
// Returns a list of maps: [[id, failedStageName, result, errors]]
@NonCPS
List < Map > getStageResults(RunWrapper build) {

  // Get all pipeline nodes that represent stages
  def visitor = new PipelineNodeGraphVisitor(build.rawBuild)
  def stages = visitor.pipelineNodes.findAll {
    it.type == FlowNodeWrapper.NodeType.STAGE
  }

  return stages.collect {
    stage ->

      // Get all the errors from the stage
      def errorActions = stage.getPipelineActions(ErrorAction)
    def errors = errorActions?.collect {
      it.error
    }.unique()

    return [
      id: stage.id,
      failedStageName: stage.displayName,
      result: "${stage.status.result}",
      errors: errors
    ]
  }
}

// Get information of all failed stages
@NonCPS
List < Map > getFailedStages(RunWrapper build) {
  return getStageResults(build).findAll {
    it.result == 'FAILURE'
  }
}

/////// ****************************** Code for fectching Failed Stage Name ******************************
///////
```

```
pipeline {
  agent any

  environment {
    deploymentName = "devsecops"
    containerName = "devsecops-container"
    serviceName = "devsecops-svc"
    imageName = "siddharth67/numeric-app:${GIT_COMMIT}"
    applicationURL = " http://devsecops-demo.eastus.cloudapp.azure.com"
    applicationURI = "/increment/99"
  }

  stages {

    stage('Testing Slack - 1') {
      steps {
        sh 'exit 0'
      }
    }

    stage('Testing Slack - Error Stage') {
      steps {
        sh 'exit 0'
      }
    }

  }

  post {
    //    always {
    //      junit 'target/surefire-reports/*.xml'
    //      jacoco execPattern: 'target/jacoco.exec'
    //      pitmutation mutationStatsFile: '**/target/pit-reports/**/mutations.xml'
    //      dependencyCheckPublisher pattern: 'target/dependency-check-report.xml'
    //      publishHTML([allowMissing: false, alwaysLinkToLastBuild: true, keepAll: true, reportDir: 'owasp-zap-report',
reportFiles: 'zap_report.html', reportName: 'OWASP ZAP HTML Report', reportTitles: 'OWASP ZAP HTML Report'])

    // //Use sendNotifications.groovy from shared library and provide current build result as parameter
    //      //sendNotification currentBuild.result
    //    }

    success {
      script {
        /* Use slackNotifier.groovy from shared library and provide current build result as parameter */
        env.failedStage = "none"
        env.emoji = ":white_check_mark: :tada: :thumbsup_all:"
        sendNotification currentBuild.result
      }
    }

    failure {
      script {
        //Fetch information about  failed stage
        def failedStages = getFailedStages(currentBuild)
        env.failedStage = failedStages.failedStageName
        env.emoji = ":x: :red_circle: :sos:"
        sendNotification currentBuild.result
      }
    }
  }

}
```

This is a Jenkins pipeline script that defines two functions for fetching information about failed stages in a Jenkins pipeline build.
The first function, getStageResults(RunWrapper build), fetches information about all stages in the pipeline, including the failure cases. It does this by using the PipelineNodeGraphVisitor class to traverse the pipeline graph and find all nodes that represent stages. For each stage node, it collects information such as the stage ID, display name, status result, and any errors associated with the stage. The function returns a list of maps, where each map contains this information for a single stage.

The second function, getFailedStages(RunWrapper build), uses the getStageResults function to fetch information about all stages and then filters this list to only include the stages that have a status result of 'FAILURE'. This function returns a list of maps, where each map contains information about a single failed stage, including the stage ID, display name, and any errors associated with the stage.
The script also includes an import statement for a Jenkins shared library called 'slack', and it uses the @NonCPS annotation to indicate that the two functions should not be checkpointed and should be executed entirely in memory. Additionally, the script includes some import statements for various Jenkins and pipeline classes that are needed for the getStageResults function.

Within the post block, there is a failure section, which is executed only if the pipeline build has failed.
Within the failure section, there is a script block, which contains some code for sending a notification about the failed build.
The code first calls the getFailedStages function, which was defined in a previous part of the pipeline script, to fetch information about the failed stages in the pipeline build. It then sets two environment variables: failedStage, which is set to the name of the first failed stage, and emoji, which is set to a string of emojis to be included in the notification message.
Finally, the sendNotification function is called with the currentBuild.result as its argument, which sends a notification message with the result of the pipeline build (i.e., "FAILURE") and includes the failedStage and emoji variables in the message.
Overall, this post block and the getFailedStages function are part of a larger pipeline script that is designed to send notifications when a pipeline build fails, and to include information about the failed stages in the notification message.

BING:
The code has two main parts:

- A @Library annotation that imports a shared library called 'slack', which presumably contains some functions or variables related to sending notifications to Slack.
- A custom function called getFailedStages that takes a RunWrapper object as an argument and returns a list of maps containing information about all the failed stages in the pipeline run. The function uses some classes and methods from the io.jenkins.blueocean.rest.impl.pipeline package, which provides an API for accessing pipeline data. The function iterates over all the pipeline nodes that represent stages, and collects their id, name, result, and errors into a list.

The purpose of this function is probably to report the failed stages to Slack or some other channel, using the imported library. The function is annotated with @NonCPS, which means it is not subject to the CPS (Continuation Passing Style) transformation that Jenkins applies to pipeline scripts. This allows the function to use some Groovy features that are not compatible with CPS, such as closures and collections.

I hope this helps you understand the code better. If you want to learn more about Jenkins pipeline code, you can check out these resources:

- Pipeline as Code: https://www.jenkins.io/doc/book/pipeline-as-code/
- Pipeline: https://www.jenkins.io/doc/book/pipeline/
- Scripted Pipeline Syntax: https://www.jenkins.io/doc/book/pipeline/syntax/#scripted-pipeline
- Blue Ocean Pipeline REST API: https://github.com/jenkinsci/blueocean-plugin/tree/master/blueocean-rest

Source: Conversation with Bing, 4/20/2023
(1) Pipeline as Code. https://www.jenkins.io/doc/book/pipeline-as-code/.
(2) Pipeline. https://www.jenkins.io/doc/book/pipeline/.

BING
The code starts with a @Library annotation, which imports a shared library called 'slack'. This library provides some functions for sending notifications to Slack channels. Then it defines a function called getStageResults, which takes a RunWrapper object as an argument and returns a list of maps containing information about all the stages in the pipeline, such as their ids, names, results, and errors. This function uses some classes from the io.jenkins.blueocean package, which are part of the Blue Ocean plugin that provides a modern user interface for Jenkins pipelines[3].

Next, the code defines another function called getFailedStages, which takes a RunWrapper object as an argument and returns a list of maps containing information about only the failed stages in the pipeline. This function calls the getStageResults function and filters out the stages that have a result of 'FAILURE'.

Then the code begins the declarative pipeline block with the keyword pipeline. Inside this block, there are several sections and directives that define the behavior and structure of the pipeline. The first section is agent, which specifies where the entire pipeline or a specific stage will execute in the Jenkins environment. In this case, the agent is set to any, which means that any available agent can run the pipeline.

The next section is post, which defines some actions to take after the pipeline or a stage has completed. In this case, there are two post conditions: success and failure. The success condition executes when the pipeline or stage has completed successfully, and the failure condition executes when the pipeline or stage has failed. Inside each condition, there is a script block that contains some Groovy code to execute. The code uses some environment variables such as currentBuild and env to access information about the current build and its environment.

The code inside the success condition sets the env.failedStage variable to "none" and the env.emoji variable to some emoji symbols that indicate success. Then it calls the sendNotification function from the slack library and passes the currentBuild.result as an argument. This function sends a notification to a Slack channel with some information about the build result and status.

The code inside the failure condition calls the getFailedStages function and assigns its return value to a variable called failedStages. Then it sets the env.failedStage variable to the name of the failed stage and the env.emoji variable to some emoji symbols that indicate failure. Then it calls the sendNotification function from the slack library and passes the currentBuild.result as an argument. This function sends a notification to a Slack channel with some information about the build result and status.

The code does not have any stages section, which is where normally one would define the stages of the pipeline and their steps. This means that this code is meant to be used as a shared library or a template for other pipelines that define their own stages.

Source: Conversation with Bing, 4/20/2023
(3) Getting Started with Pipelines. https://www.jenkins.io/pipeline/getting-started-pipelines/.

GOOGLE BARD
The code you provided is a Jenkins pipeline that can be used to notify Slack when a build fails. The pipeline first imports the following libraries:

- slack: This library is used to send notifications to Slack.
- io.jenkins.blueocean.rest.impl.pipeline.PipelineNodeGraphVisitor: This library is used to get information about the stages of a pipeline.
- io.jenkins.blueocean.rest.impl.pipeline.FlowNodeWrapper: This library is used to represent a stage in a pipeline.
- org.jenkinsci.plugins.workflow.support.steps.build.RunWrapper: This library is used to represent a build in Jenkins.
- org.jenkinsci.plugins.workflow.actions.ErrorAction: This library is used to represent an error action in a pipeline.

The pipeline then defines a function called getStageResults(). This function takes one argument, build, which is a RunWrapper object that represents the build. The function returns a list of maps. Each map in the list represents a stage in the build. The maps have the following properties:

- id: The ID of the stage.
- failedStageName: The name of the failed stage.
- result: The result of the stage.
- errors: A list of errors that occurred in the stage.

The pipeline then defines a function called getFailedStages(). This function takes one argument, build, which is a RunWrapper object that represents the build. The function returns a list of maps. Each map in the list represents a failed stage in the build. The maps have the same properties as the maps returned by the getStageResults() function.

The script then defines a function called getFailedStages(). This function takes one argument, build, which is a RunWrapper object. The function simply calls the getStageResults() function and passes it the build argument. The function then filters the results to only include stages with a result of FAILURE.

The script finally calls the slackSend() function to send a notification to Slack. The function passes the following arguments to the slackSend() function:

- color: The color of the notification
- title: The title of the notification
- text: The text of the notification
- attachments: The attachments for the notification

The slackSend() function is a Groovy function that is defined in the slack library. The function sends a notification to Slack. The notification can have a color, title, text, and attachments.

The pipeline then defines a post section. This section is executed when the build is finished. The post section has two branches: a success branch and a failure branch.

The success branch is executed when the build succeeds. The success branch does the following:

- Sets the environment variable failedStage to the empty string.
- Sets the environment variable emoji to a list of emojis that indicate success.
- Calls the sendNotification() function with the current build's result as the argument.

The failure branch is executed when the build fails. The failure branch does the following:

- Gets a list of failed stages from the getFailedStages() function.
- Sets the environment variable failedStage to the name of the first failed stage.
- Sets the environment variable emoji to a list of emojis that indicate failure.
- Calls the sendNotification() function with the current build's result as the argument.

The sendNotification() function is a function that is defined in the slack library. This function takes one argument, result, which is the result of the build. The function sends a notification to Slack that contains the following information:

- The name of the build.
- The result of the build.
- The name of the failed stage, if the build failed.
- A list of emojis that indicate the result of the build.

The following is an example of a notification that is sent to Slack when a build succeeds:

```
Build `my-build` has succeeded!
```

The following is an example of a notification that is sent to Slack when a build fails:

```
Build `my-build` has failed!
The failed stage is `deploy`.
```

thumb_upthumb_downrefreshGoogle it

more_vert

From < https://bard.google.com/u/1/>