Art of the Feature Toggle

Patterns for maintaining and improving Shiny applications over time
Solutions Engineering at RStudio
Chapter 12 - DevOps Philosophy and Tooling

- “Production” has many meanings
- Are there lessons can be learned from the DevOps community?
Introducing new features over time
Shiny Applications in Production
Adding new features to Shiny in Production

Version 1

Version 2
Adding new features to Shiny in Production
Adding new features to Shiny in Production

Version 1

Version 2
Does pushing a code change feel risky?
DevOps Learning: Decouple deployment from release

- **Deployment** is any push of code to an environment (test, prod)
- **Release** is when that code (feature) is made available to users or customers

Deployment on demand and thoughtful release strategies allow more control (and more success) over the delivery of features to end users.

- Application-based release patterns (today!)
- Environment-based release patterns (tomorrow)
Application-Based Release Patterns
Feature Toggle

A mechanism to selectively enable and disable features, or control which features are visible to specific user segments.

Enable **dark launch**: deploy a change to production and then perform testing while it’s invisible to most users.

- Modify the code to make calls to new functions, log results without displaying
- Have 1% of users make invisible calls to new feature, fix issues, progressively increase users to test production load
Starting from the Gallery

Professional Features
These examples demonstrate some of the unique features of RStudio Connect.

- **Airline Delays for American Airlines**
- **Sales Reports**

**Authentication and database**

**Personalized UI**

Smells like feature toggles!
"Personalized Data Access" session$user

Sales Reports

Monthly Manager Sales Report

September Sales Projections

<table>
<thead>
<tr>
<th>salesperson</th>
<th>day</th>
<th>dailySales</th>
</tr>
</thead>
<tbody>
<tr>
<td>sales1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>sales1</td>
<td>1</td>
<td>758.85</td>
</tr>
</tbody>
</table>

Sales Reports

Monthly Sales Report for 'sales1'

September Sales Projections

<table>
<thead>
<tr>
<th>salesperson</th>
<th>day</th>
<th>dailySales</th>
<th>salesTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>sales1</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>sales1</td>
<td>1</td>
<td>758.85</td>
<td>758.85</td>
</tr>
</tbody>
</table>
Feature Toggles

Shiny applications can access the username and groups of the current user through the session parameter of the `shinyServer` function.

Your application could use this information to display customized messages or to enable functionality for a specific subset of users.

```r
shinyServer(function(input, output, session) {
  output$username <- reactive({
    session$user
  })

  output$groups <- reactive({
    session$groups
  })
})
```
Implementation

```r
# Render the subtitle of the page according to what user is logged in.
output$subtitle <- renderText({
  if (is.null(user())){
    return("You must log in to use this application using Shiny Server Pro."
  }

  if (isManager()){
    return("Monthly Manager Sales Report")
  } else {
    return(paste0("Monthly Sales Report for ", user(), ")")
  }
})
```
Implementation

UI built through conditional session$user tests within server-side output objects

```r
output$salesTbl <- renderDataTable({
  # If no user is logged in, don't show any data.
  if (is.null(user())){
    return()
  }
  
  # Otherwise return all data that should be visible to this user.
  myData()
})
```

output_subtitle  output$salesPlot  output$salesTbl
Dynamic UI Feature Toggles
The **conditionalPanel** function

The **conditionalPanel** function, is used in `ui.R` and wraps a set of UI elements that need to be dynamically shown/hidden.

Creates a panel that shows and hides its contents depending on the value of a JavaScript expression, usually input-based.

This example shows how to define an output variable in the server code that you can use in the UI.

```r
library(shiny)

ui <- fluidPage(
  selectInput("num", "Choose a number", 1:10),
  conditionalPanel(
    condition = "output.square",
    "That's a perfect square!"
  )
)

server <- function(input, output, session) {
  output$square <- reactive({
    sqrt(as.numeric(input$num)) %% 1 == 0
  })
  outputOptions(output, 'square', suspendWhenHidden = FALSE)
}

shinyApp(ui = ui, server = server)
```

**Output condition example:** Dean Attali
output$isBeta <- reactive({
  [Test for group membership]
})

conditionalPanel(
  condition = "output.isBeta == false",
  ... UI Elements)

conditionalPanel(
  condition = "output.isBeta == true",
  ... UI Elements)
Feature Toggles in Production?

Will I (personally) use feature toggles for Shiny application deployment and releases? **No, not likely.**

- Not easy to manage, automate, test and ultimately “toggle”
  - Alternative: “Environment-Based Release Patterns”

- Still a cool concept, useful for solving other types of problems
  - RStudio Connect Feature Hacks!
Mixing Access Controls & Feature Toggles on RStudio Connect
## Access Control Options on RStudio Connect

### Access Control Setting

<table>
<thead>
<tr>
<th>Access Control Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anyone, no login required</td>
<td>Allows for anonymous viewer access</td>
</tr>
<tr>
<td>All users, login required</td>
<td>Everyone with an RStudio Connect account</td>
</tr>
<tr>
<td>Specific users or groups</td>
<td>Only users or groups you specify can access</td>
</tr>
<tr>
<td>Collaborators &amp; you</td>
<td>Only you and your collaborators can access</td>
</tr>
</tbody>
</table>
People > Group Management UI in RStudio Connect (Admin Tools)

Solutions Engineering group setup
Problem: Can you add me as a collaborator on this?

1. **Self service code access**
   - `install.packages("gitlink")`

2. **Slack Incoming Webhook integration**

   If `session$groups != solutions` {
   
   Display an action button that will send me a slack alert when clicked.
   }

---

Who can view this application

- All users - login required

Who can change this application

- [k] kelly
- [s] solutions

Add collaborator
“Slack me if you need access”

Problem: Can you add me as a collaborator on this?

1. **Self service code access**
   
   ```r
   - install.packages("gitlink")
   ```

2. Slack Incoming Webhook integration

   If `session$groups != solutions` {
   
   Display an action button that will send me a slack alert when clicked.
   }

   ```r
   Add collaborator
   ```
“Slack me if you need (collaborator) access”

```
conditionalPanel(
    condition = "output.isCollab == false",
    actionButton("request", "Request Collaborator Access")
)
```

colorado.rstudio.com says
"Your Collaboration Request Has Been Sent."

[Slack and email icons]
Action Button - Command Pattern

Pattern 1 - Command

Use `observeEvent()` to trigger a command with an action button.

Example

```r
library(shiny)

ui <- fluidPage(
  tags$head(tags$script(src = "message-handler.js"),
  actionButton("do", "Click Me")
)

server <- function(input, output, session) {
  observeEvent(input$do, {
    session$sendCustomMessage(type = 'testmessage',
      message = 'Thank you for clicking')
  })
}

shinyApp(ui, server)
```

Request Collaborator Access

```
colorado.rstudio.com says

"Your Collaboration Request Has Been Sent."
```

OK
Recommendation: **Create a Slack App**

There are a number of R packages and legacy methods you could use to talk to slack from your R code. I recommend creating a Slack App.

Send data to Slack in real-time

- **Incoming Webhook**

Creating an Incoming Webhook gives you a unique URL to which you send a **JSON** payload with the message text and some options.
Add features and functionality

Choose and configure the tools you'll need to create your app (or review all our documentation).

- **Incoming Webhooks**
  Post messages from external sources into Slack.

- **Interactive Components**
  Add buttons to your app's messages, and create an interactive experience for users.

- **Slash Commands**
  Allow users to perform app actions by typing commands in Slack.

- **Event Subscriptions**
  Make it easy for your app to respond to activity in Slack.

- **Bots**
  Add a bot to allow users to exchange messages with your app.

- **Permissions**
  Configure permissions to allow your app to interact with the Slack API.

Install your app to your workspace

Install your app to your Slack workspace to test your app and generate the tokens you need to interact with the Slack API. You will be asked to authorize this app after clicking **Install App to Workspace**.

App Name

Connect Collaborate Notifier

Display Information

This information will be shown in the Slack App Directory and in the Slack App. For more information, view our App Detail Guidelines.

<table>
<thead>
<tr>
<th>App name</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect Collaborate</td>
<td>A Webhook for Collaboration Requests on RStudio Connect</td>
</tr>
</tbody>
</table>

App icon & Preview

![App icon](image)
“Slack _me_ if you need access”

Mentioning users

A mention is a special type of reference that will provide a link to the mentioned user's profile, and also notify them about the reference. The help center page for mentions describes what that notification process looks like.

To mention a user in an app-published message, you need to provide their user ID in the following syntax:

Hey <@U024BE7LH>, thanks for submitting your report.

- Find your slack member ID
- Construct your message
- Send JSON payload to Webhook URL
Action Button - Command Pattern

```
conditionalPanel(
    condition = "output.isCollab == false",
    actionButton("request", "Request Collaborator Access")
)
```

```
observeEvent(input$request, {
    collab_alert <- glue('{{"text":"Hey {{slack_owner_id}}, {{session$user} is requesting collaborator access to {{vanity_url}}"}}}')
    POST(slack_webhook, body = collab_alert, add_headers('Content-Type' = 'application/json'))
    session$sendCustomMessage(type = 'confirm-request',
        message = 'Your Collaboration Request Has Been Sent.')
})
```

```
// This receives messages of type "confirm-request" from the server.
Shiny.addCustomMessageHandler("confirm-request",
    function(message) {
        alert(JSON.stringify(message));
    }
);
```
Sales Reports

Monthly Sales Report

Request Collaborator Access

September Sales Projections

Access Control Setting | Group or User Conditional | Solution
---|---|---
All users, login required | Group isSolutions() FALSE | Element for requesting collaborator access

Connect Collaborate Notifier

Hey @kelly, rsc_admin is requesting collaborator access to https://colorado.rstudio.com/rsc/collab-notify/
Left: Publisher view with access controls on RStudio Connect
Signed in as Kelly, who is a member of solutions

Right: Vanity URL app access by a different user
User is not a member of the solutions group

Request Access button displayed
### More Hacks for `session$user`

Create a custom landing page for anonymous visitors and logged out users

<table>
<thead>
<tr>
<th>Access Control Setting</th>
<th>Group or User Conditional</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anyone, no login required</td>
<td>is.null(user()) TRUE</td>
<td>Custom UI for anonymous viewers</td>
</tr>
</tbody>
</table>
Left:
Publisher view with access controls on RStudio Connect

Right:
Custom UI for anonymous and logged-out visitors to the app

Watch on YouTube
Code: 


- solutions.rstudio.com
- community.rstudio.com
- kelly@rstudio.com