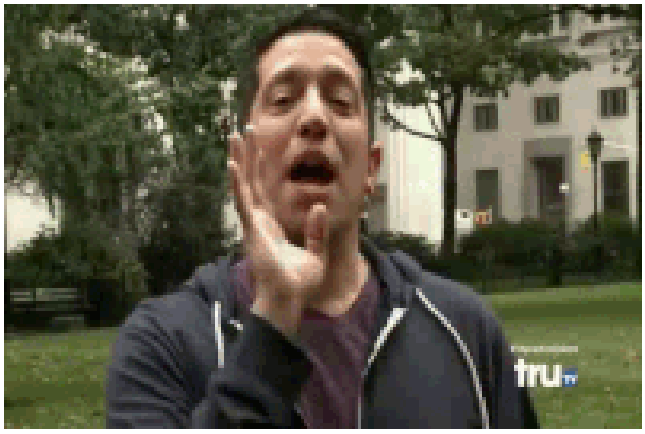
The Red Head's Tech Blog: Interactive Bots

Posted by FALYCIA HANKINSON Apr 14, 2017

Hey, hey you. Want to add buttons to your Slack bot app? Need some slash commands?

Have you already read through my other post for installing a new custom app?



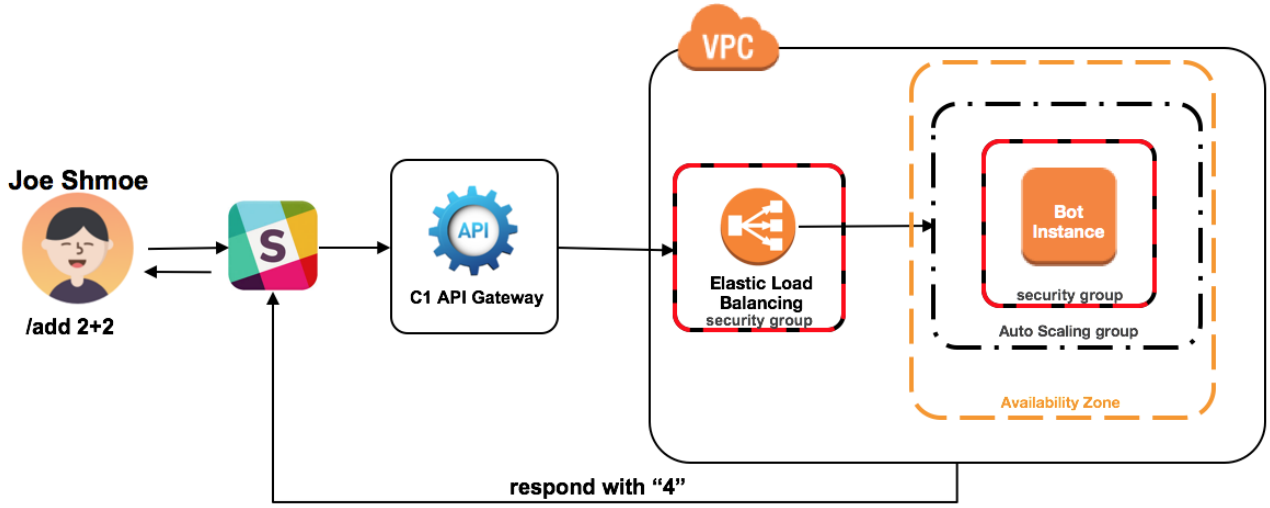
Slack has additional features that you can add to your custom application. Notice how I said custom application and not bot? That's because you can't get these additional features in a simple bot, you have to create a custom app. No way around that.

The "additional features" that I am going to be discussing in this post include Slash Commands, Event Subscriptions, and Interactive Messages.
Want to see an example Go Slackbot that uses interactive buttons? Take a peek at my woofer-interactive repo.

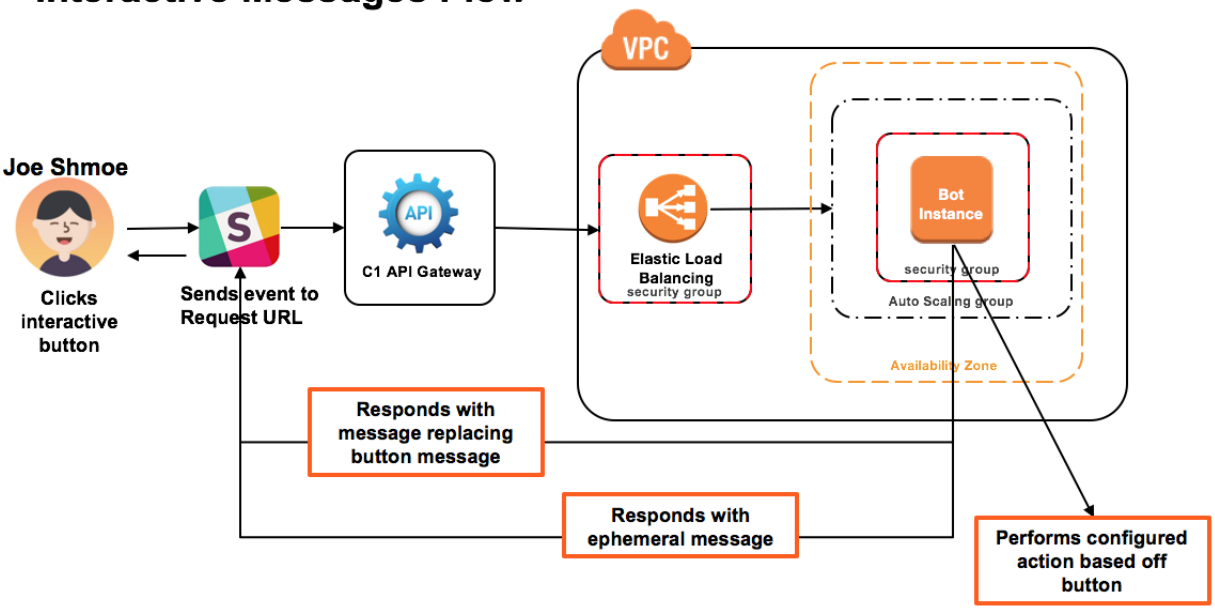
Architecture Examples

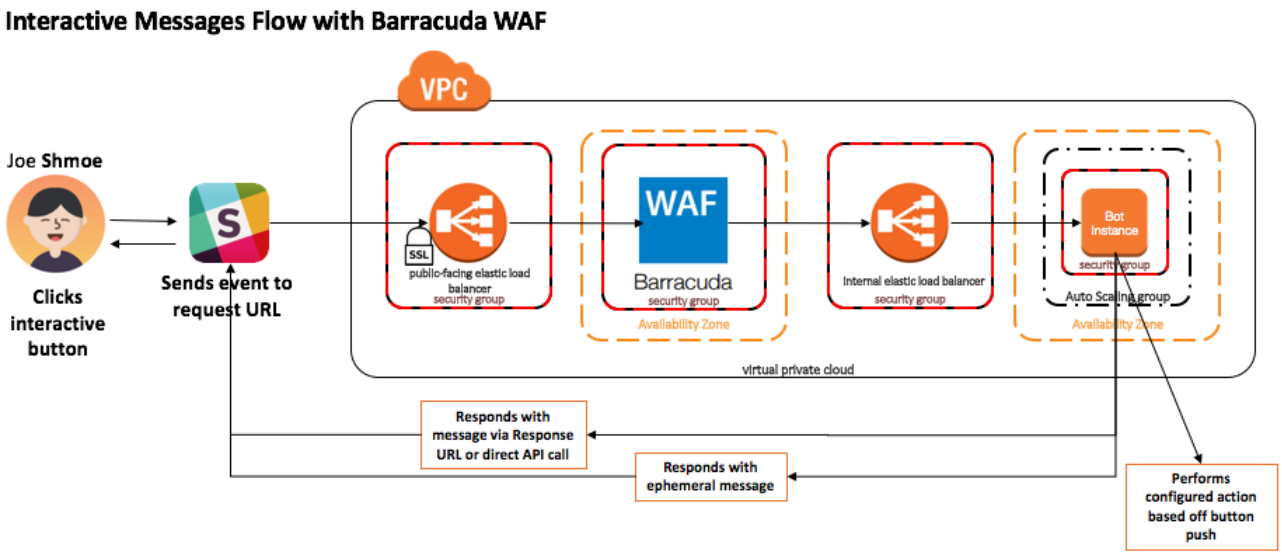
In order to enable these features on your custom application you need to host an endpoint that Slack can push data to. In layman's terms, you need to make your app publicly accessible to Slack via a URL (e.g., https://mybot.capitalone.com/slashcommands, https://mybot.capitalone.com/buttons). There are lots of pieces to this to do it right and I am sure I may leave out some roadblocks you may hit along the way due to changing policies on public facing entities, etc.

Slash Commands Flow



Interactive Messages Flow





1. YOU NEED A PUBLIC CERT

First, notice how in my example urls, I put "https"? That's because to use these features, Slack requires you use SSL with a verified and trusted certificate for them to send you data. This technically is not necessary if you use the CapOne API Gateway (if I'm not mistaken) since they already have a public-facing presence. To get a public cert, here are some helpful links: Pulse-CSR Creation and Enrollment, Symantec CSR Walkthrough, another Pulse walkthrough, and the actual Symantec enrollment link.

Simply because I like to help , here is a SUPER AWESOME WALKTHROUGH of getting the public cert and attaching it to an internet-facing ELB. (I am biased)

2. YOU NEED A PUBLIC-FACING API GW (Gateway) or WAF

Currently there are a couple solutions for this. I have tested with using the AWS API Gateway in non-production but policy dictates you have to have the AWS WAF in place to use it which is not currently approved for COF use and is still being evaluated: AWS Services Summary & Best Practices

Barracuda WAF is also available to be launched across LOB's and its advanced functionality is by far above anything we currently have as far as detection and data concealment. Unfortunately you can't put a Barracuda in front of AWS API Gateway or Lambda (just yet) but you can put in front of an internal ELB to get Slack to talk to you (see diagram above).

We have a CapOne API Gateway that allows us to create new API's as well. I personally haven't gone this route yet since the on-boarding process for the CapOne API Gateway is somewhat of a pain for just playing around with new Slackbots. Hopefully the team that manages it can see this blog post and come up with an

"express" lane for generic bots to communicate with Slack. Do some searching on Pulse since there are some tutorials I've glanced over plus had a teammate stumble across this link too.

So to reiterate:
Capital One API Gateway option:

- pros: existing public presence (valid certs)
- cons: time-consuming for initial on-boarding

Barracuda WAF option:

- pros: easy deploy and setup, manage own WAF rules
- cons: you deploy and own it, need Symantec SSL cert

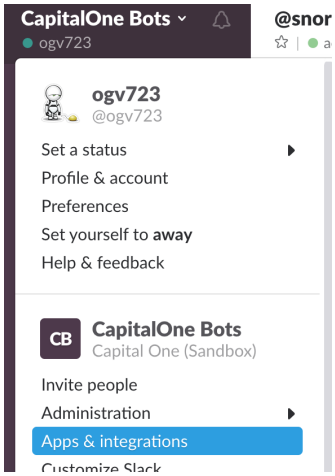
3. YOU NEED AN API SERVER (SOMETHING FOR THE API GATEWAY OR WAF TO POINT TO)

This can be an EC2 instance, an ECS ALB, a lambda, doesn't matter. You just need something that is listening for API calls and is configured to take action based off of the calls received.
A simple restful api with an http listener is sufficient. Simple Go examples here, here, and here. Go/Lambda/Gateway example here. Slack has docs on the expected data that will be sent for each type of feature. That will be discussed below.

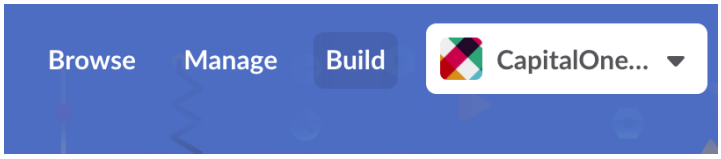
Slash Commands

1. Configuration:

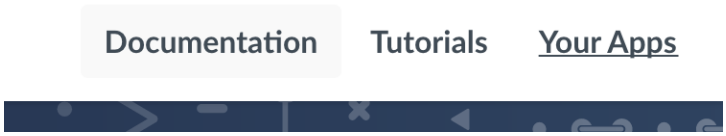
To enable Slash commands on your custom app, navigate to your app configuration by going to apps and integrations



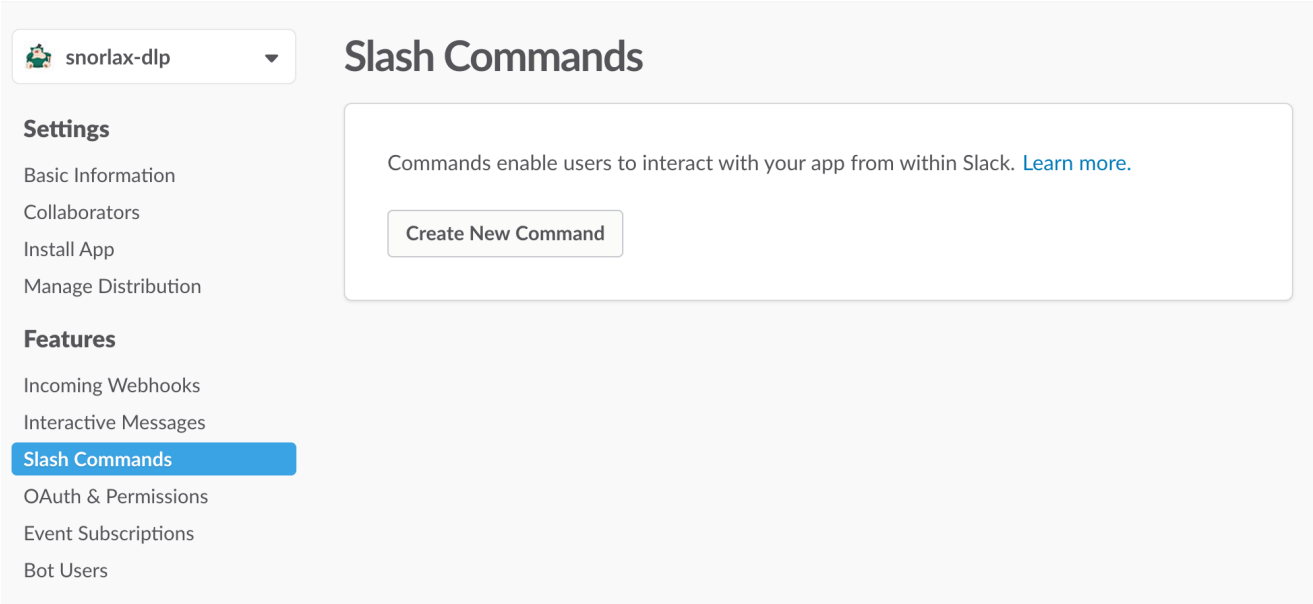
Click build at the top right



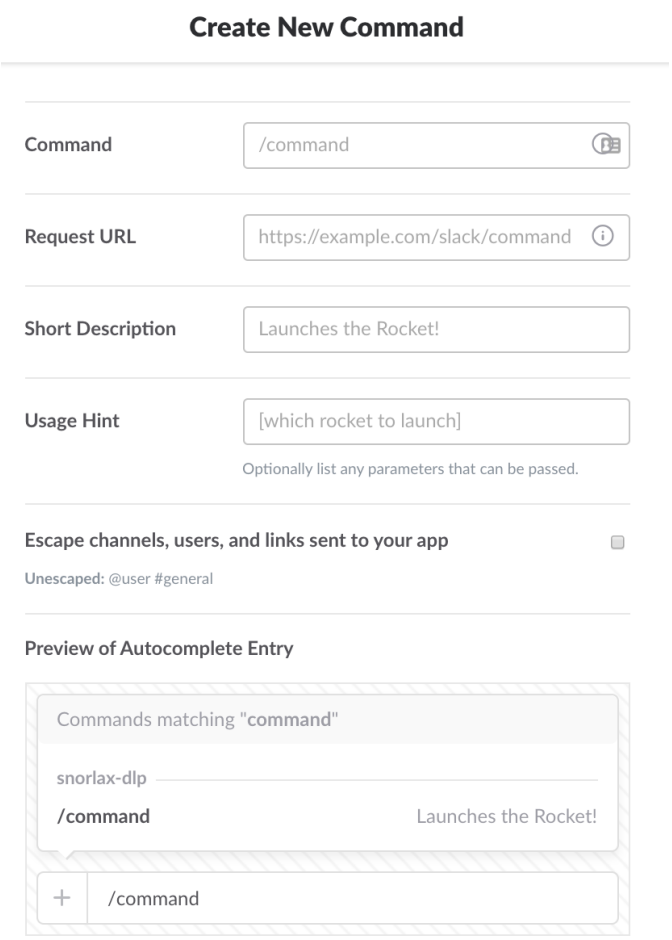
Click "Your Apps" at the top right then click your app you want to configure



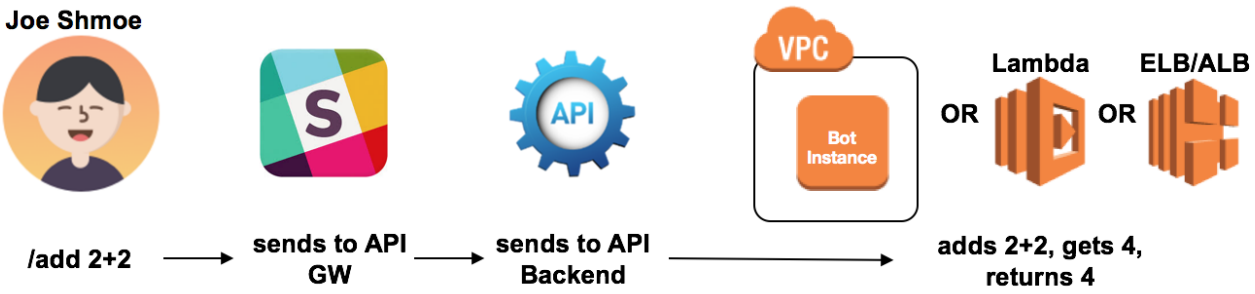
Then add the commands here:



A new command configuration looks like this:

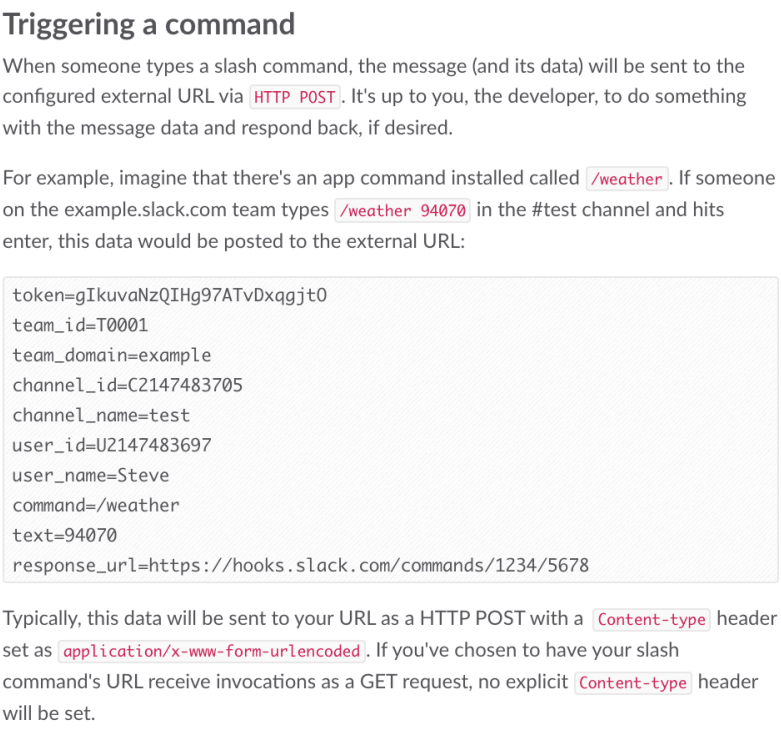


So if I wanted to allow a user to call my bot like this "/add 2+4" I would put "/add" in the command section and add my API GW endpoint url as the request URL.
After you configure this, here is what the actual flow will look like utilizing API GW. Switch out GW with WAF public-facing ELB if using a WAF:



2. What Slack will send your endpoint:

An example post that Slack gave (reference):



Triggering a command

When someone types a slash command, the message (and its data) will be sent to the configured external URL via `HTTP POST`. It's up to you, the developer, to do something with the message data and respond back, if desired.

For example, imagine that there's an app command installed called `/weather`. If someone on the example.slack.com team types `/weather 94070` in the #test channel and hits enter, this data would be posted to the external URL:

```
token=gIkuvaNzQIHg97ATvDxqgjt0
team_id=T0001
team_domain=example
channel_id=C2147483705
channel_name=test
user_id=U2147483697
user_name=Steve
command=/weather
text=94070
response_url=https://hooks.slack.com/commands/1234/5678
```

Typically, this data will be sent to your URL as a HTTP POST with a `Content-type` header set as `application/x-www-form-urlencoded`. If you've chosen to have your slash command's URL receive invocations as a GET request, no explicit `Content-type` header will be set.

I would run extensive tests to get the EXACT post data because Slack isn't exactly known for giving the most up-to-date information on what you should expect from them.
Side note: Here is an example of a bot using Lambda, Python, and slash commands we were playing around with:GitHub - ISRM-Security-Engineering/artybot .

Make sure to read the entire Slack post on triggering slash commands because it has a lot of good points on validating data following the slash, dealing with userid's, etc.
3. Responding to what Slack sends you:
Same reference, different section (reference)

Slack allows you to do simple plain-text responses back to them when you just have a simple answer.
But more complex responses require a set JSON layout for Slack to translate the output:
{ "text": "It's 80 degrees right now.", "attachments": [{ "text":"Partly cloudy today and tomorrow" }] }

"Your URL should respond with a HTTP 200 "OK" status code. Any other flavor of response will result in a user-facing error.
If you don't want to respond with any message or would rather send your responses later using delayed responses, you should still respond to the invocation of your URL with a simple HTTP 200."
If you don't want to send a response but rather trigger an action elsewhere, just send them back a status 200 to let them know you received their message. Then you can trigger a separate function call to complete a task based off of the slash command you received.

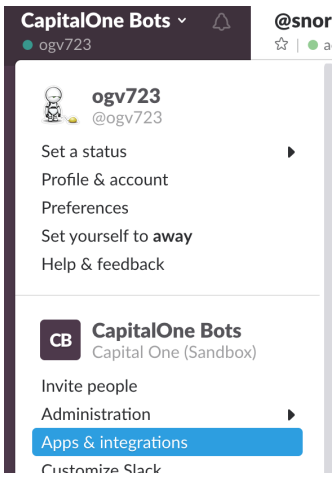
Interactive Messages

Must. Have. Buttons!!! (reference)
GIF stolen from Slack.

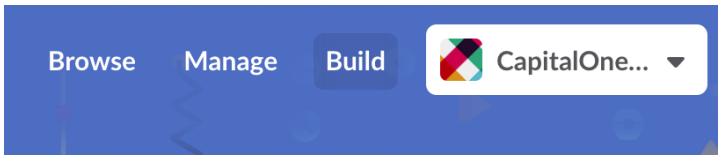
Interactive messages allow users to "interact" with your message... See what I did there?

1. Configuration:

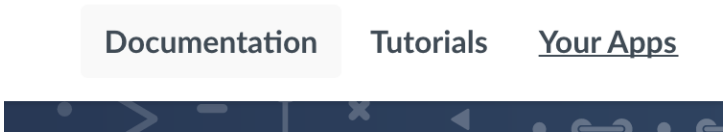
To enable Interactive Messages on your custom app, navigate to your app configuration by going to apps and integrations



Click build at the top right



Click "Your Apps" at the top right then click your app you want to configure



Click Interactive Messages on the left side bar then enable interactive messages. Then, you need to configure a Request URL (Slack sends button push events to this url). The Options Load URL is for messages that include Menus. Not going down that rabbit hole right now.

2. Process Flow:

Slack really did a good job at breaking down the flow process of interactive messages:

1. Your application produces a message containing actions. Maybe it has buttons. Maybe it has menus. Maybe it has both! Now all the users who can read it can interact with it.
2. Users encounter your message and, inspired by its call to action, clicks or makes a selection. This triggers an invocation of your application's associated Action URL.
3. Slack sends a request to your Action URL, sending all the context needed to identify the originating message, the user executing the action, and the specific values you've associated with the action. This request also contains a response_url you can use to continue interacting with the user or channel.
4. Your app responds to the action. If responding directly to the incoming invocation request, your provided message will replace the existing message. Or respond with an ephemeral message, visible only to the invoking user. Or respond even more simply with just HTTP 200 OK and delay continuing the interaction until later using the provided response_url. There are many ways for your app to respond.
5. Meanwhile: Your application does whatever it does as a result of the intended action to be taken: enqueue a process, save a database row or some prince or princess captured in a castle. All the while your app may continue interacting with users through additional messages and evolving actions.
6. By using the response_url, your app can continue interacting with users up to 5 times within 30 minutes of the action invocation. Use this to continue through a workflow until it is complete.
7. Messages can evolve. By using chat.update and your created message's message_ts field, you can modify the original interactive message (including all of its attachments) to add or remove buttons or menus based on user interactions.

But if you like pictures:

Dabbling with creating messages with buttons in Go

1079 Views Tags: slackbot, slackbot programming, slack api, interactive messages, slash commands

 Abishek Bhaskaran
Mar 21, 2018 7:50 PM
hello , i m trying to use the slash commands to run a jenkins job . I have configured the slash commands and i m already receiving notication from Jenkins in my slack channel .But when i try to send the slash command its not working . any help would be highly appreciated . thanks !

 FALYCIA HANKINSON *in response to Devin Shields on page 10*
Nov 1, 2017 3:24 PM
For a single room in a single workspace? No. I would recommend using a combination of an RTM bot utilizing @mentions for the bot, so instead of a slash command:

/command do the stuff

the "command" would be:

@botname do the stuff

 Devin Shields
Nov 1, 2017 5:50 AM
hi! do you think all the public API setup work is plausible / worth it for slash commands in a single room / targeted at a specific team? I'm thinking about chatops.