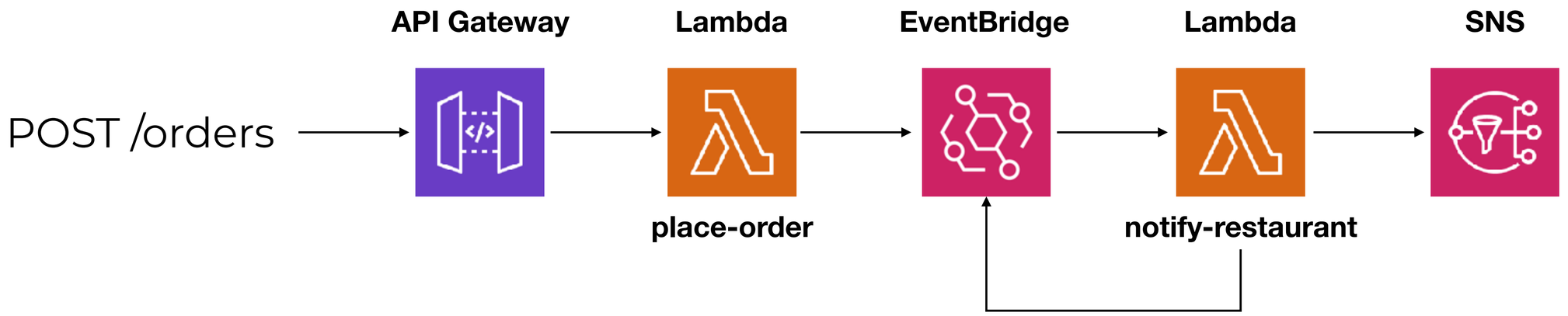
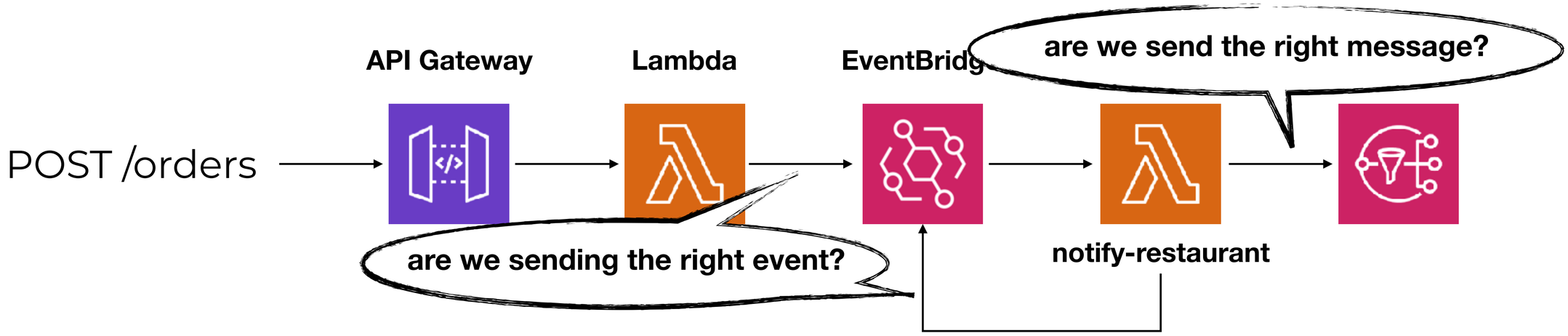
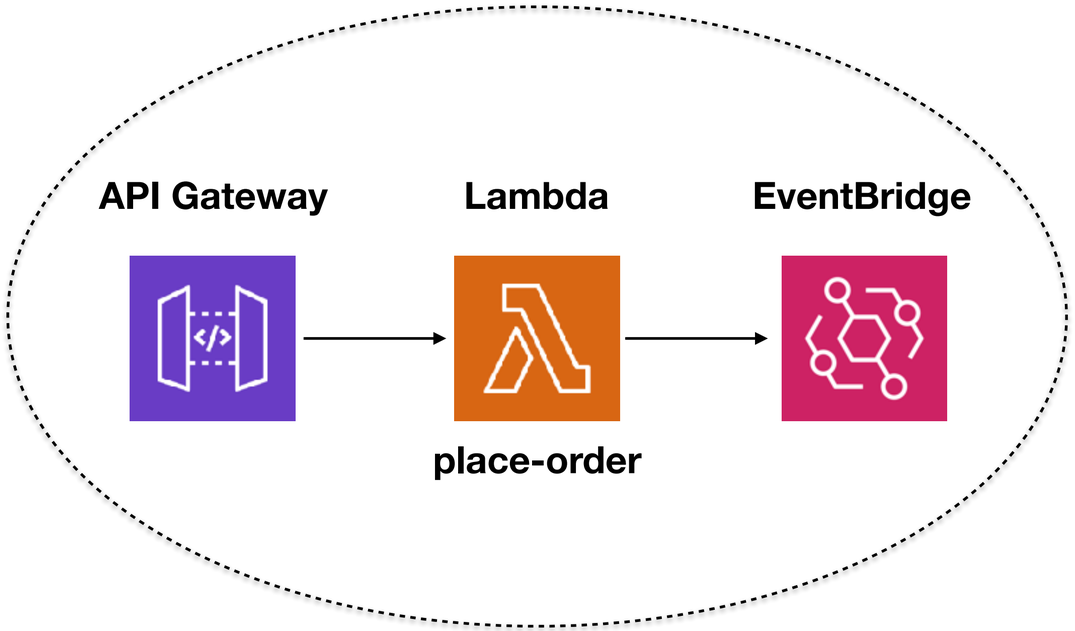
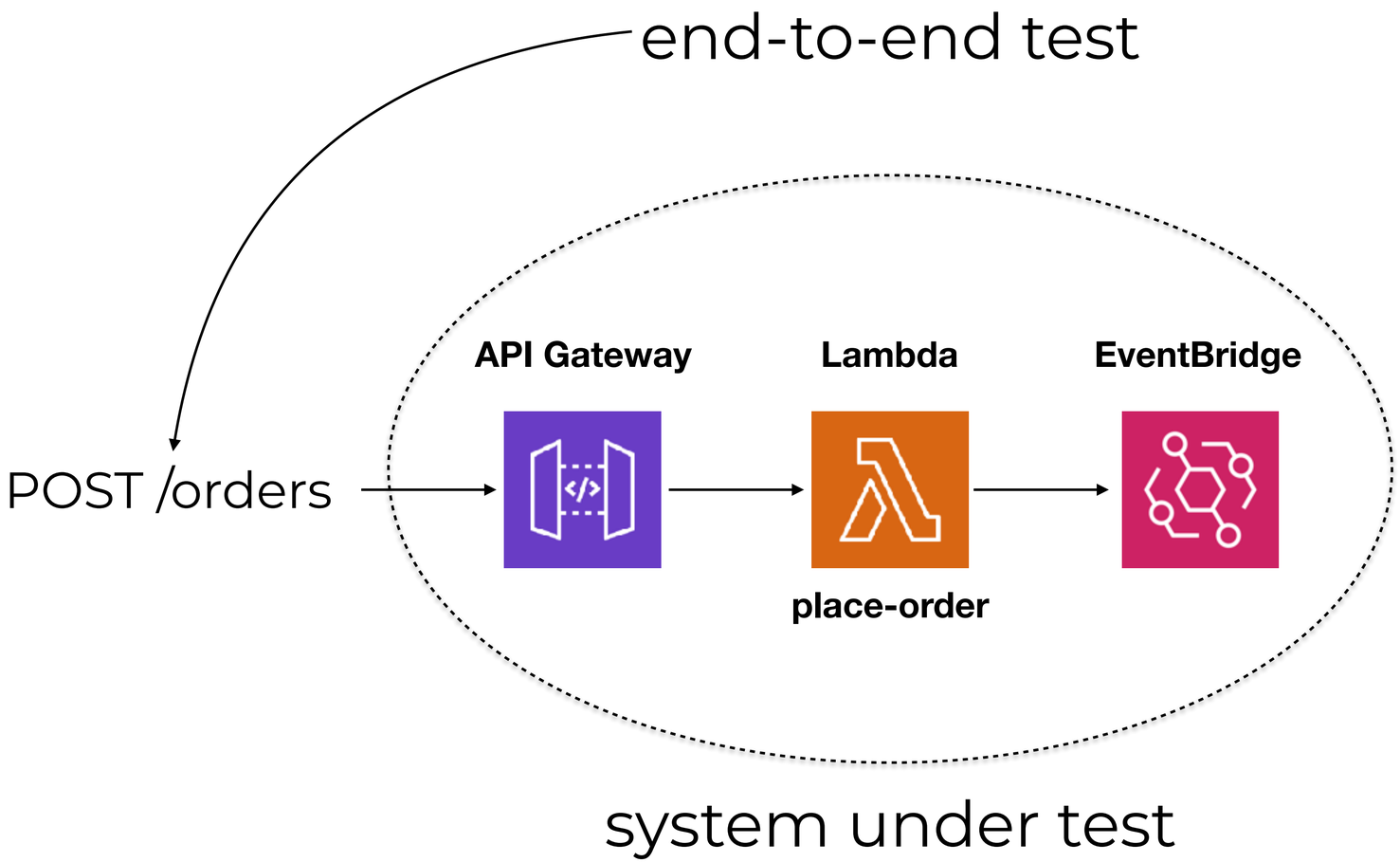

API Gateway
Lambda
EventBridge
Lambda
SNS
POST /orders
place-order
notify-restaurant

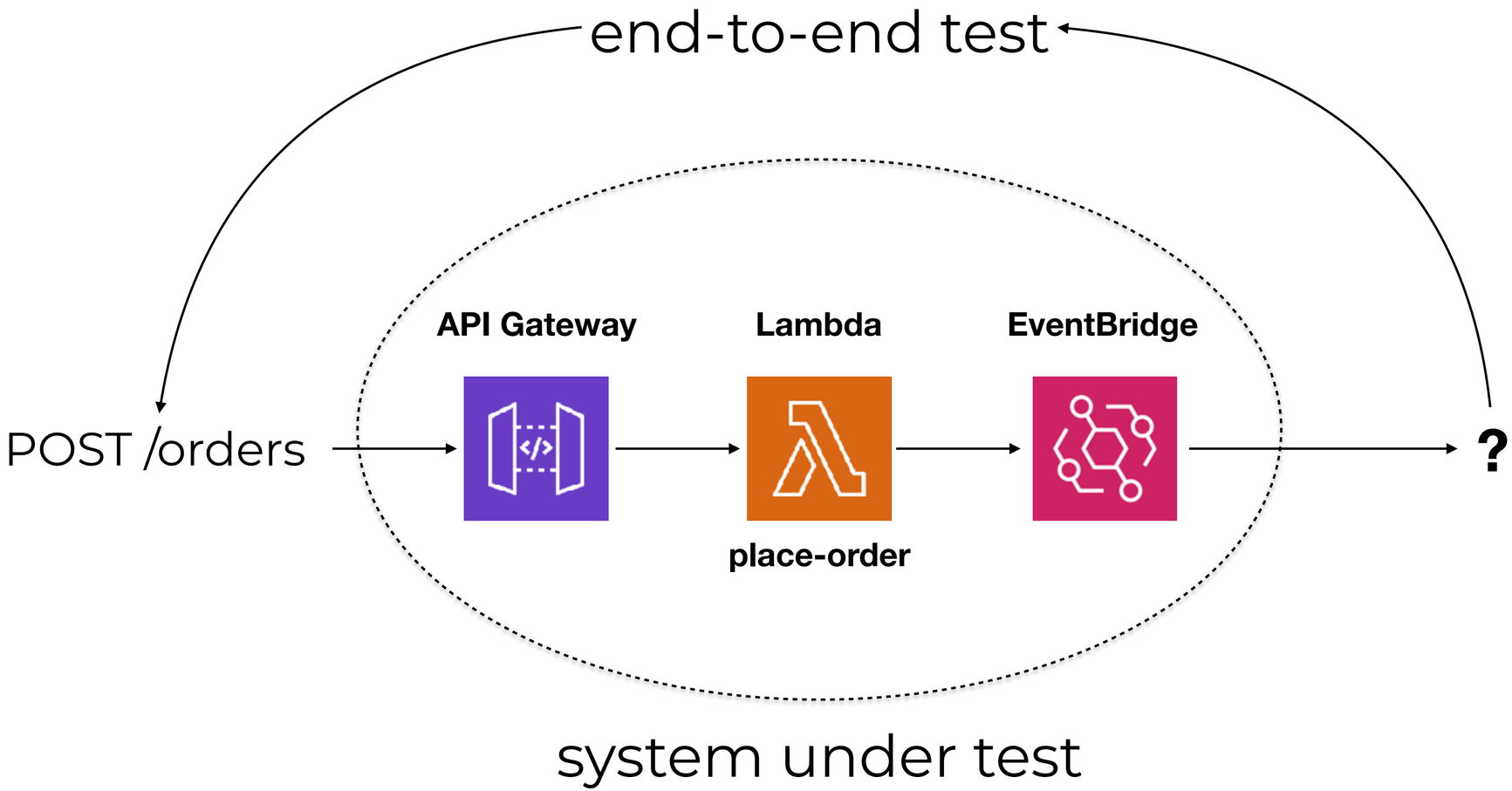

POST /orders
API Gateway
Lambda
EventBridge
are we send the right message?
are we sending the right event?
notify-restaurant

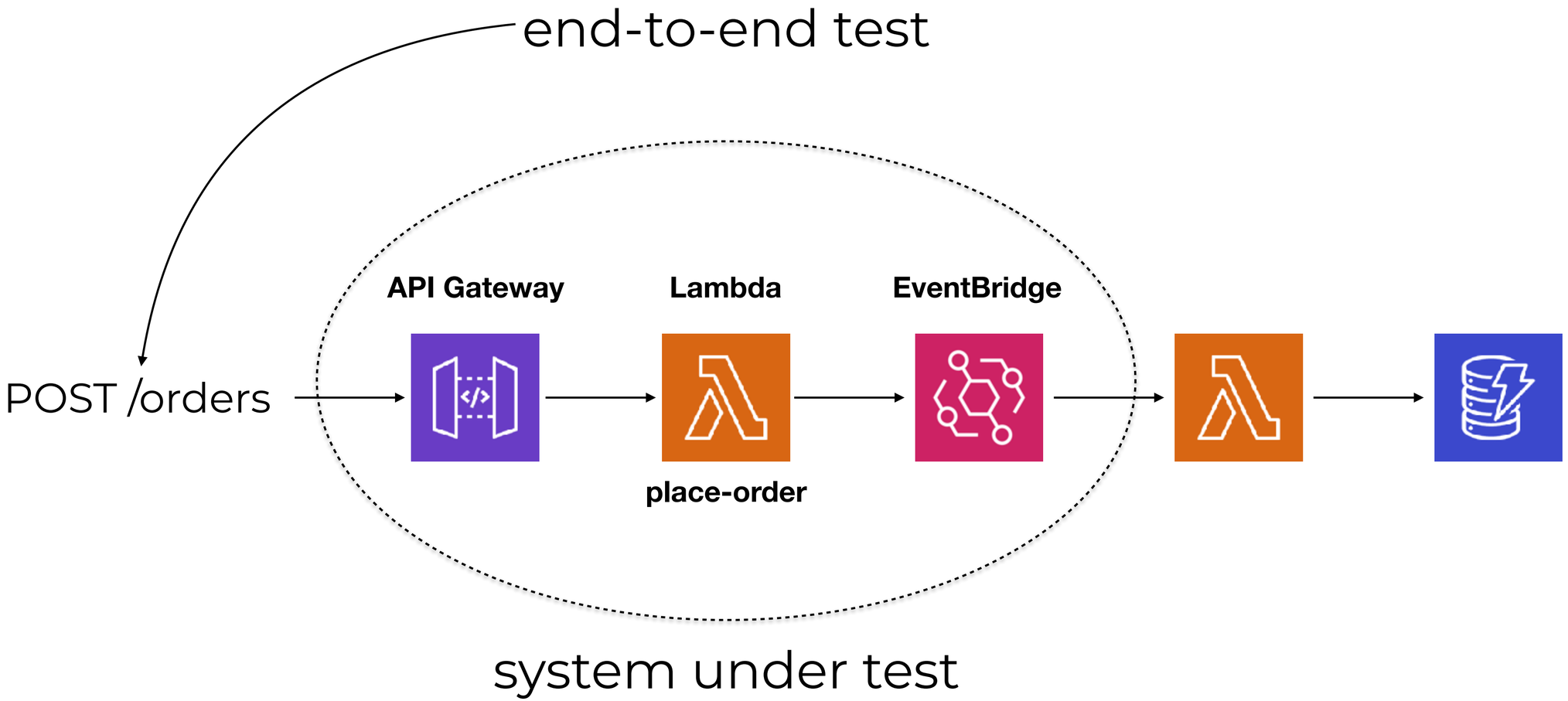
need a way to “listen in” on what’s poublished
to EventBridge and SNS

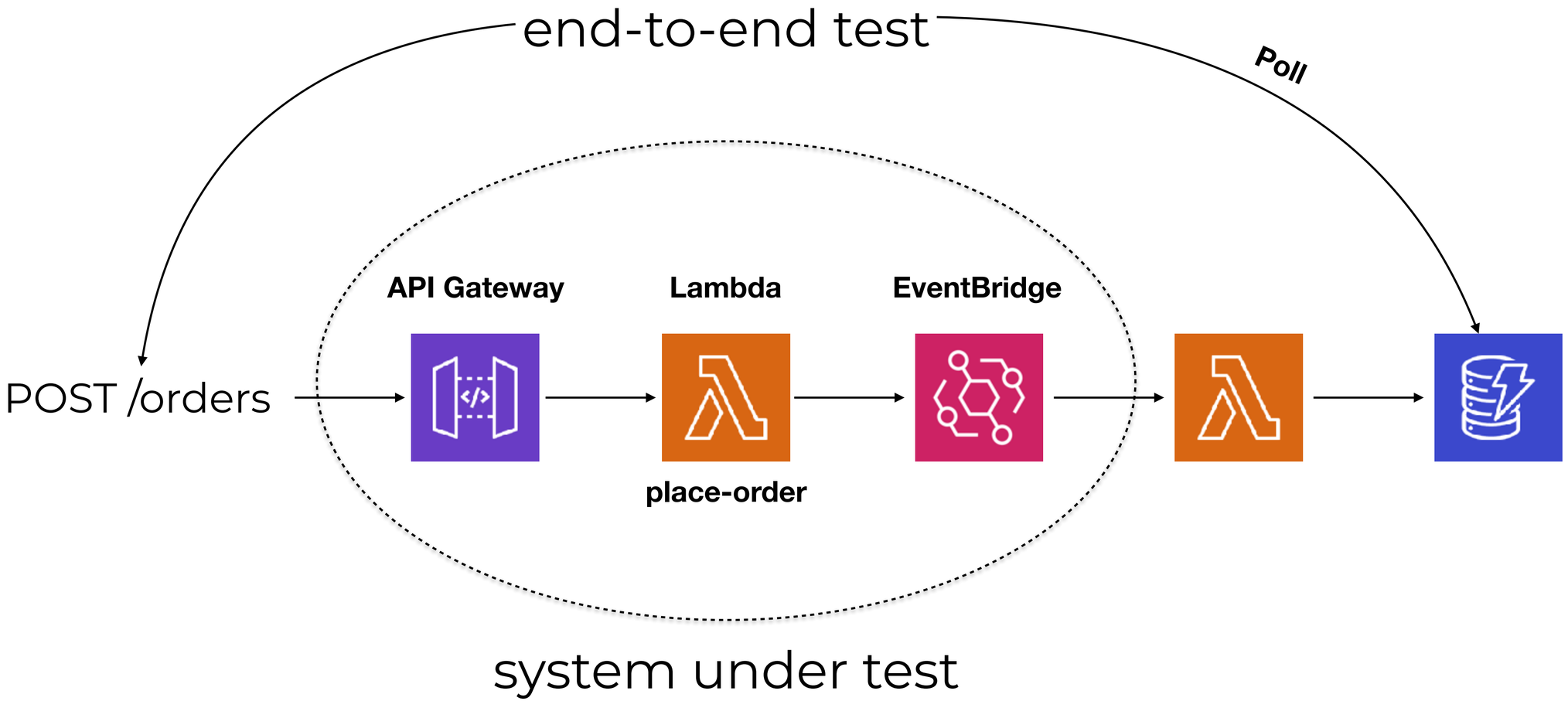


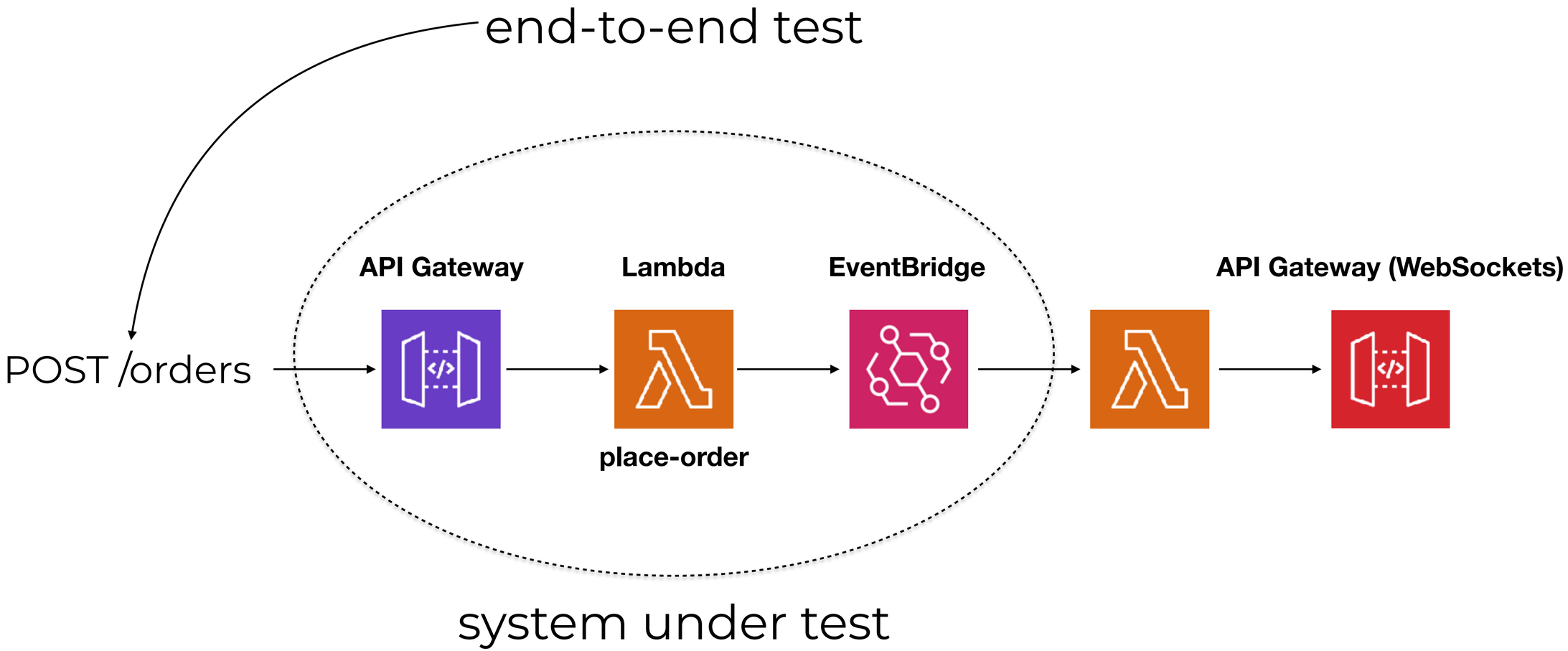
system under test

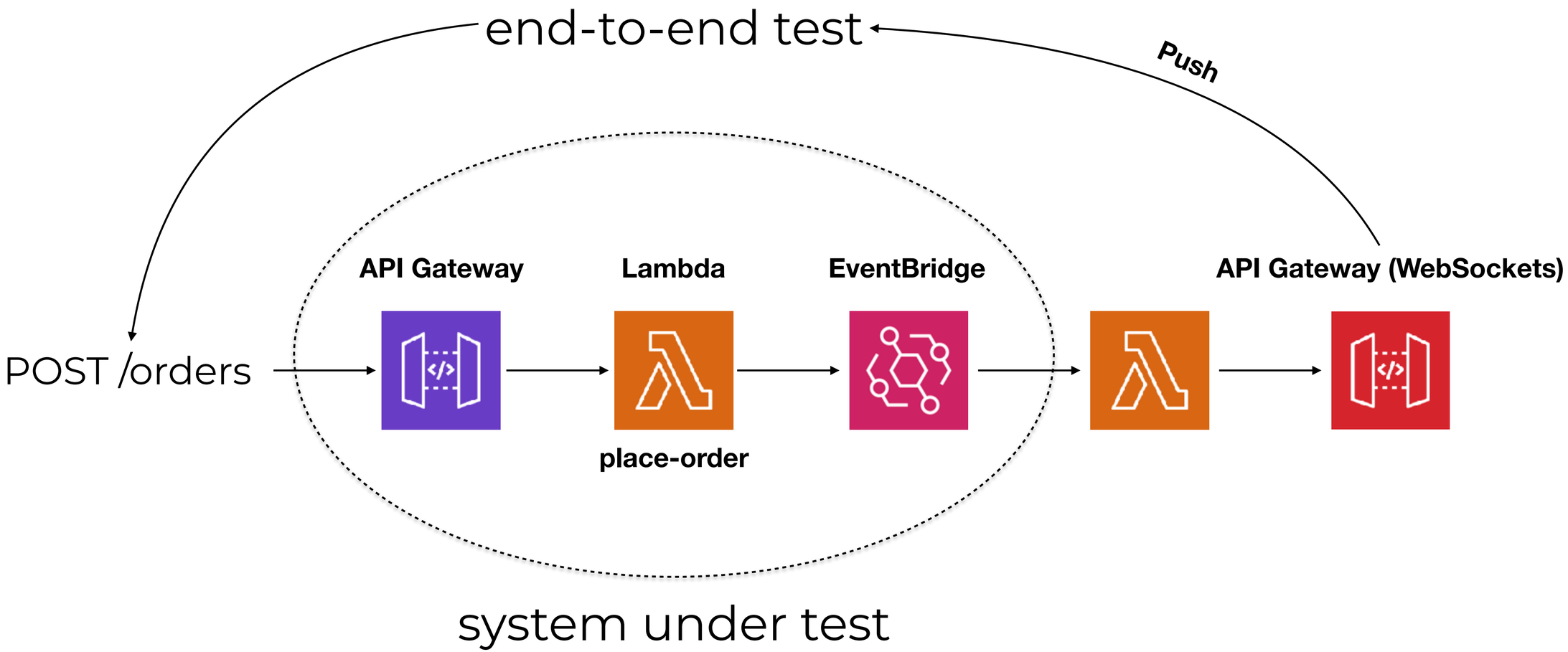

end-to-end test
POST /orders
API Gateway
Lambda
EventBridge
place-order
system under test

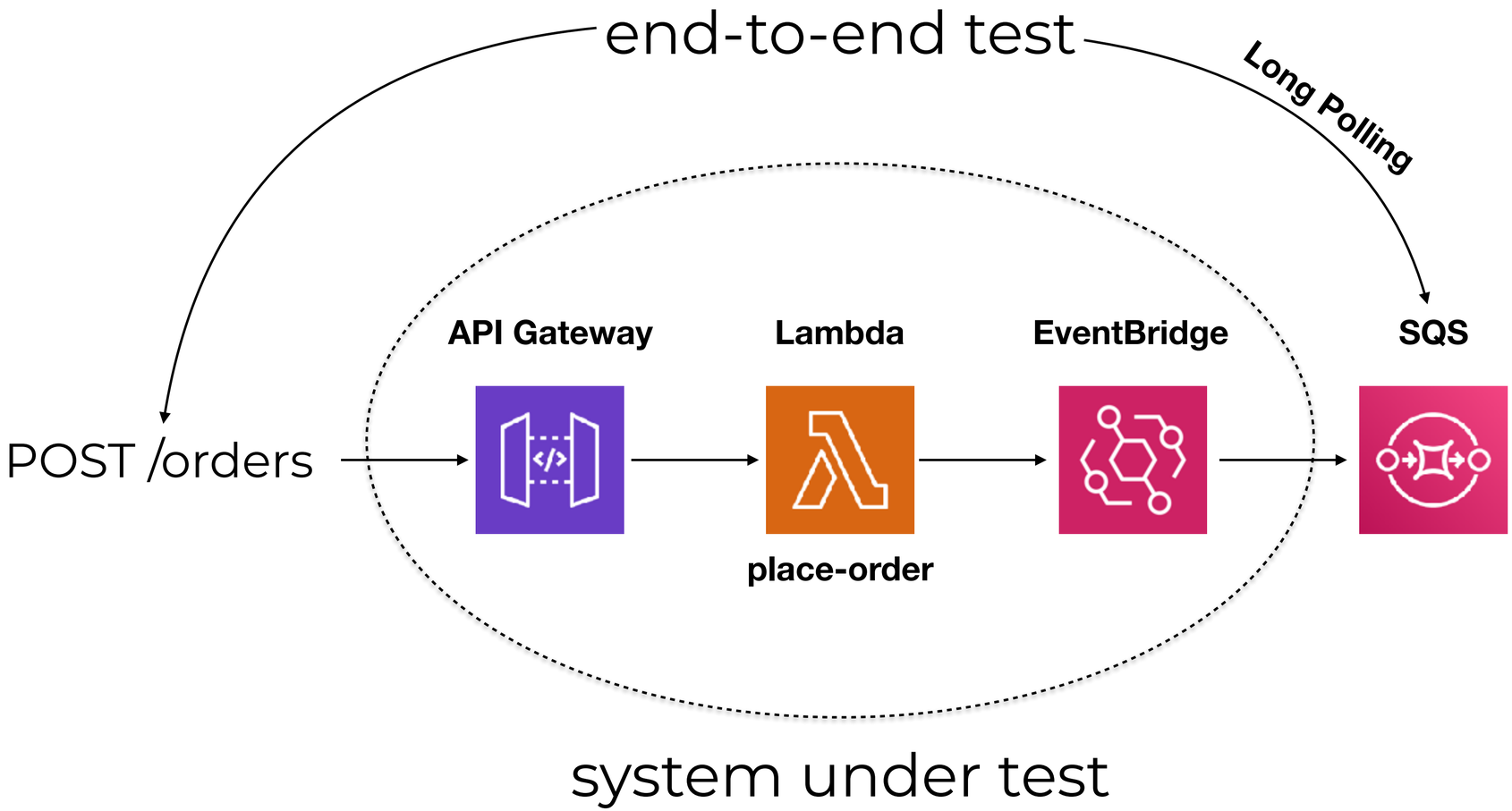

end-to-end test
API Gateway
Lambda
EventBridge
POST /orders
?
place-order
system under test

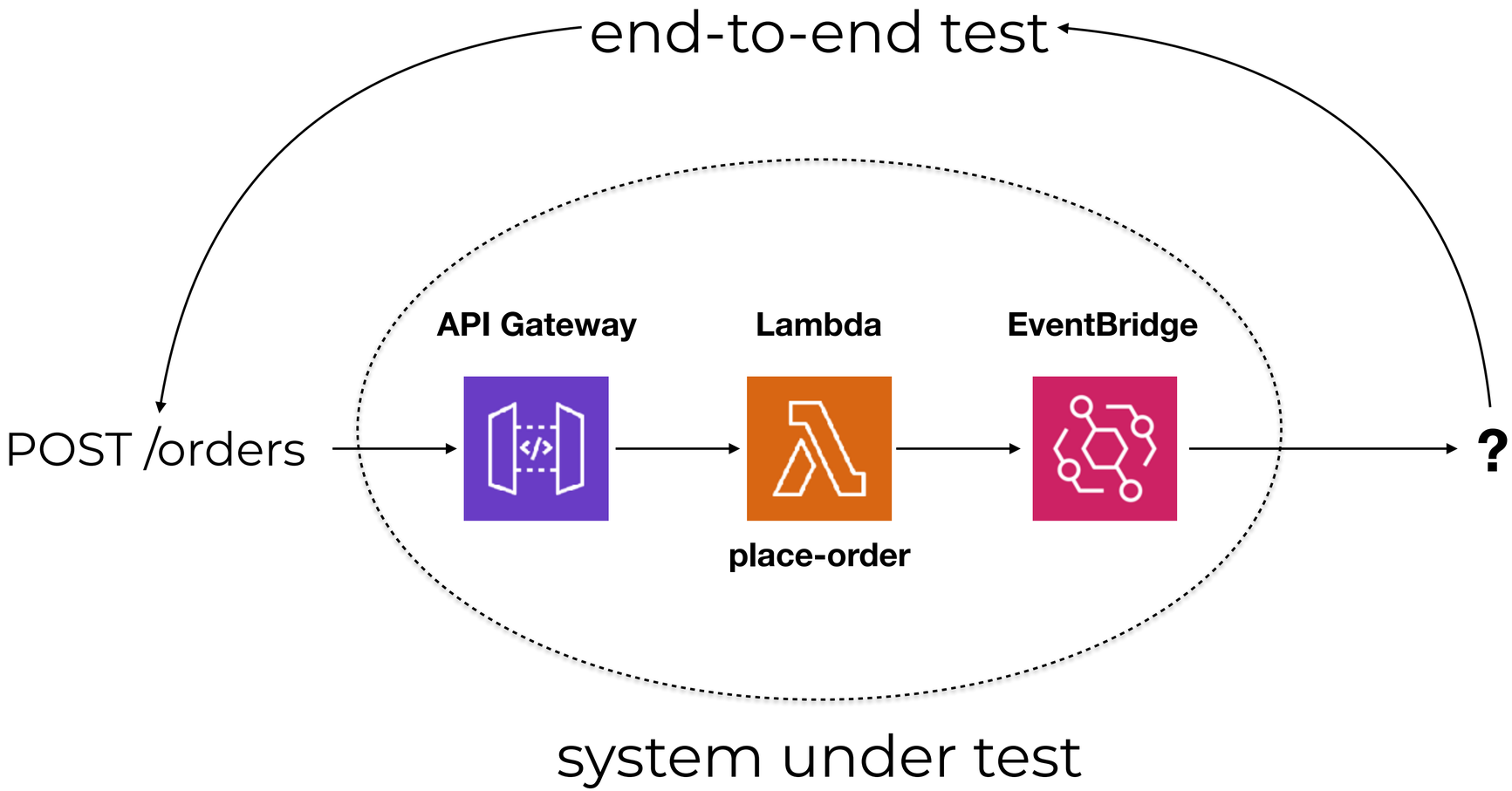

end-to-end test
POST /orders
API Gateway
Lambda
EventBridge
place-order
system under test

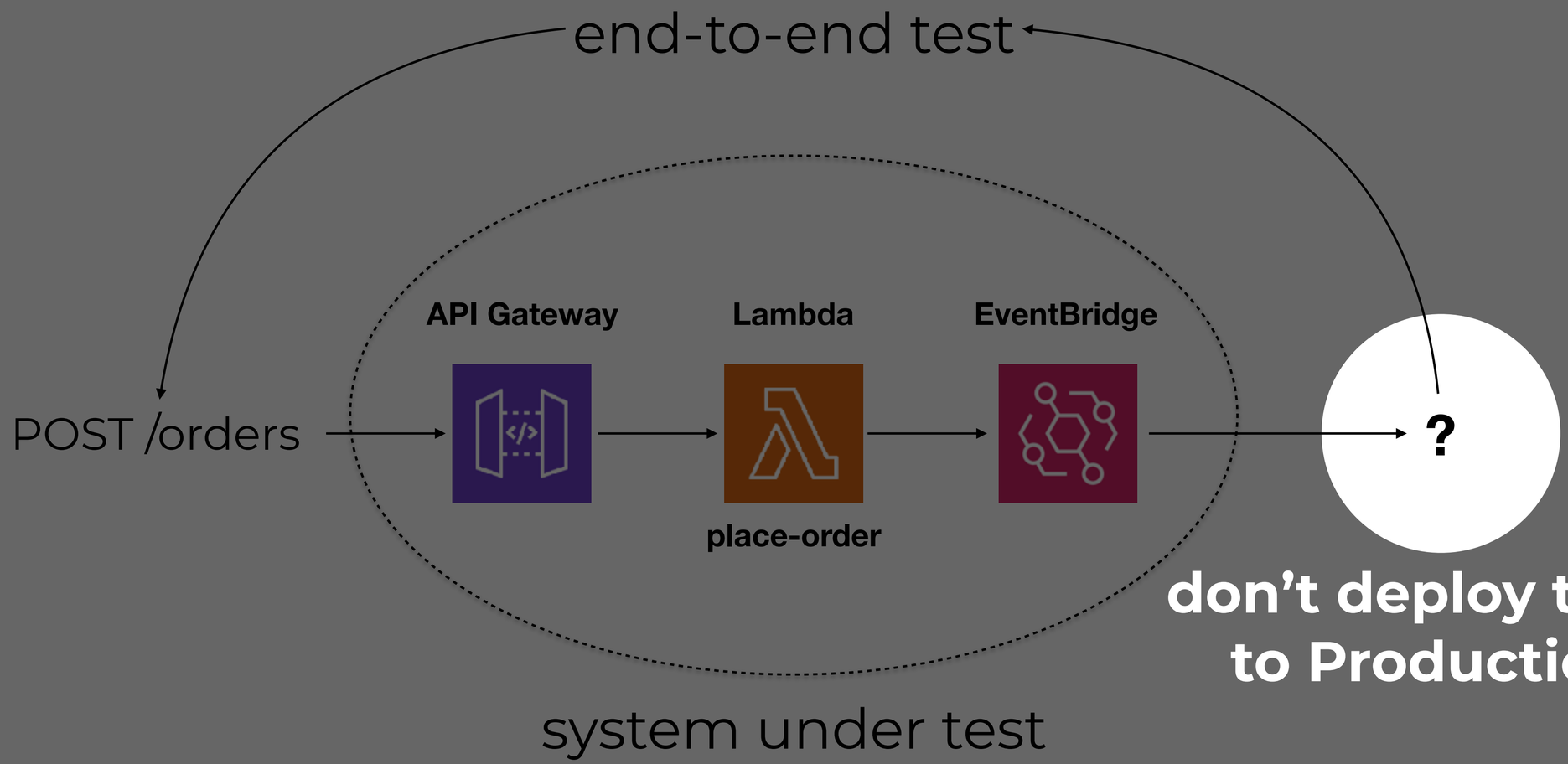

end-to-end test
Poll
POST /orders
API Gateway
Lambda
EventBridge
place-order
system under test


end-to-end test
API Gateway
Lambda
EventBridge
API Gateway (WebSockets)
POST /orders
place-order
system under test


end-to-end test
Push
POST /orders
API Gateway
Lambda
EventBridge
API Gateway (WebSockets)
place-order
system under test


end-to-end test
Long Polling
API Gateway
Lambda
EventBridge
SQS
POST /orders
place-order
system under test


end-to-end test
API Gateway
Lambda
EventBridge
POST /orders
?
place-order
system under test



don’t deploy these to Production!

Conditions

PDF | Kindle | RSS

Filter View All

The optional `Conditions` section contains statements that define the circumstances under which entities are created or configured. For example, you can create a condition and then associate it with a resource or output so that AWS CloudFormation only creates the resource or output if the condition is true. Similarly, you can associate the condition with a property so that AWS CloudFormation only sets the property to a specific value if the condition is true. If the condition is false, AWS CloudFormation sets the property to a different value that you specify.

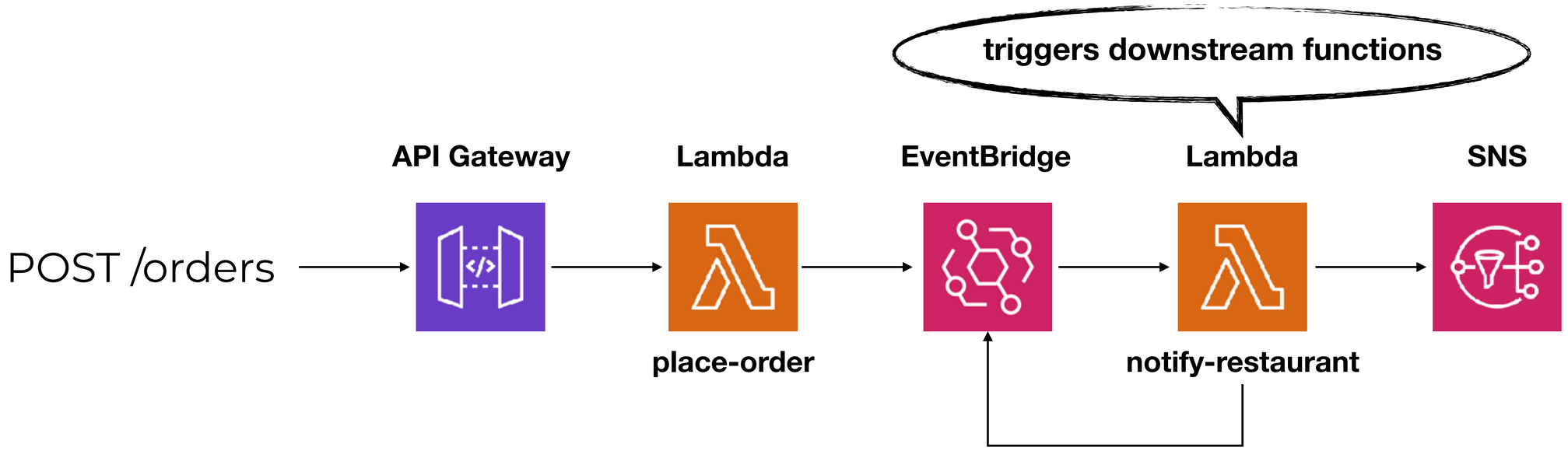
You might use conditions when you want to reuse a template that can create resources in different contexts, such as a test environment versus a production environment. In your template, you can add an `EnvironmentType` input parameter, which accepts either `prod` or `test` as inputs. For the production environment, you might include Amazon EC2 instances with certain capabilities; however, for the test environment, you want to use reduced capabilities to save money. With conditions, you can define which resources are created and how they're configured for each environment type.

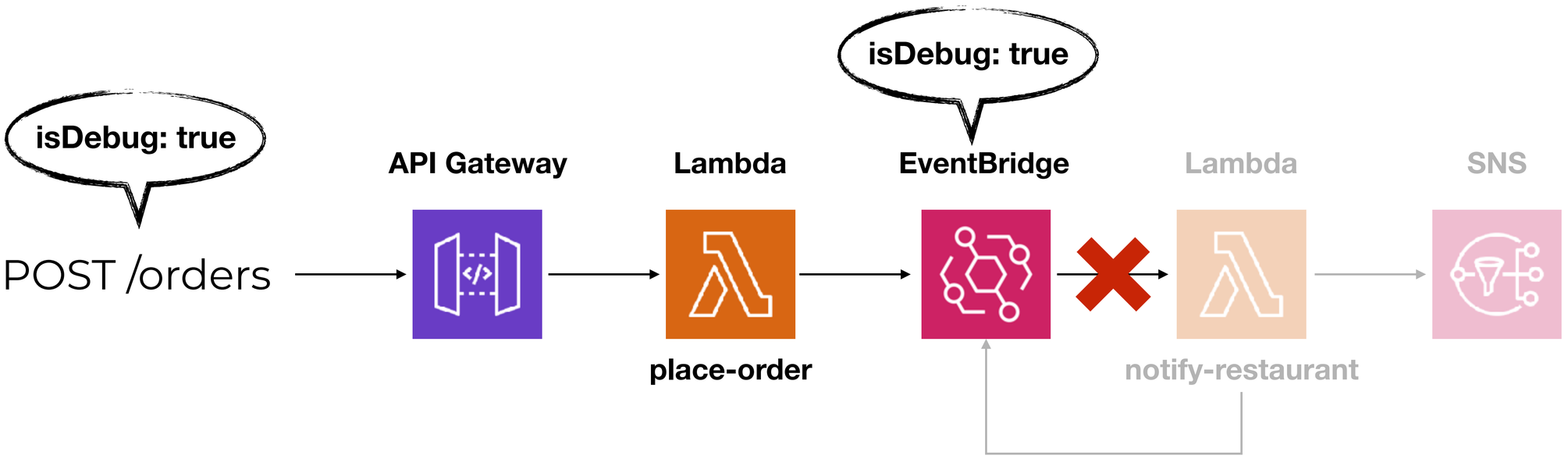
Conditions are evaluated based on predefined pseudo parameters or input parameter values that you specify when you create or update a stack. Within each condition, you can reference another condition, a parameter value, or a mapping. After you define all your conditions, you can associate them with resources and resource properties in the `Resources` and `Outputs` sections of a template.

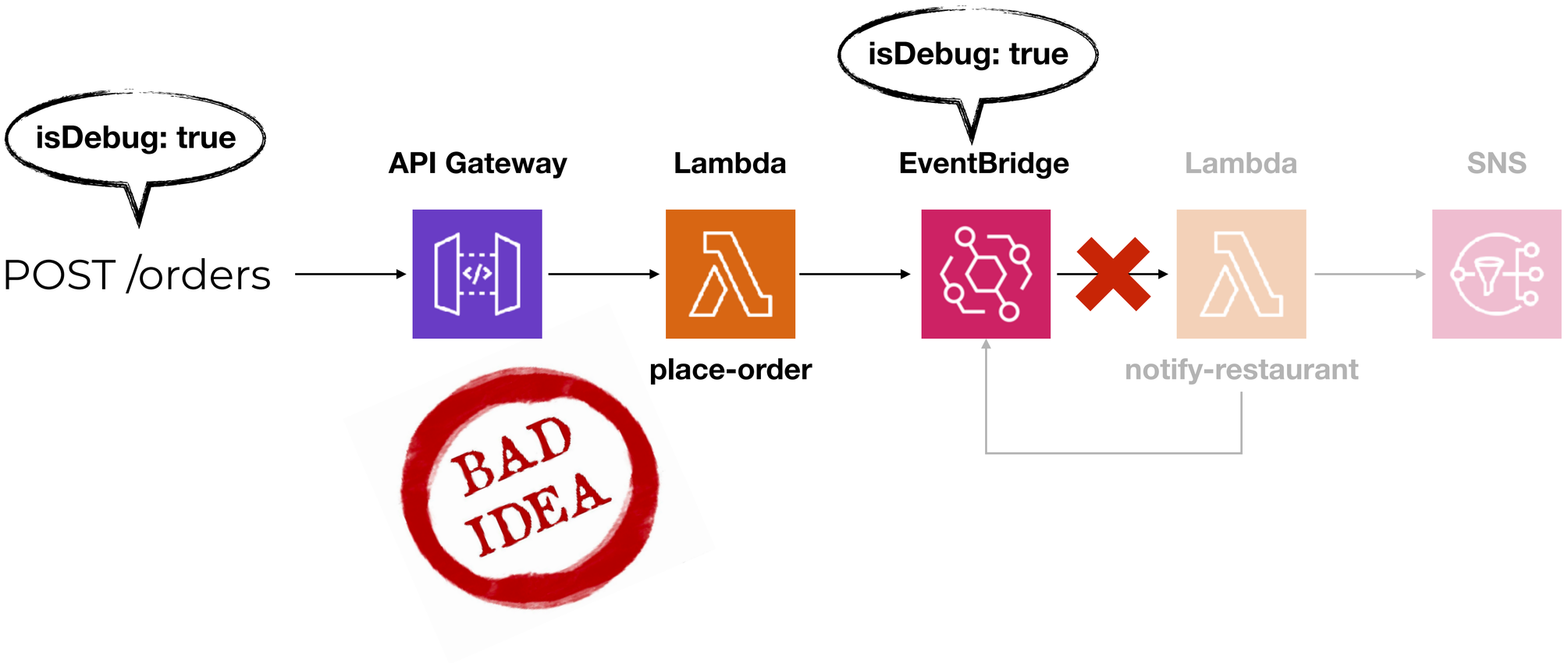
At stack creation or stack update, AWS CloudFormation evaluates all the conditions in your template before creating any resources. Resources that are associated with a true condition are created. Resources that are associated with a false condition are ignored. AWS CloudFormation also re-evaluates these conditions at each stack update before updating any resources. Resources that are still associated with a true condition are updated. Resources that are now associated with a false condition are deleted.

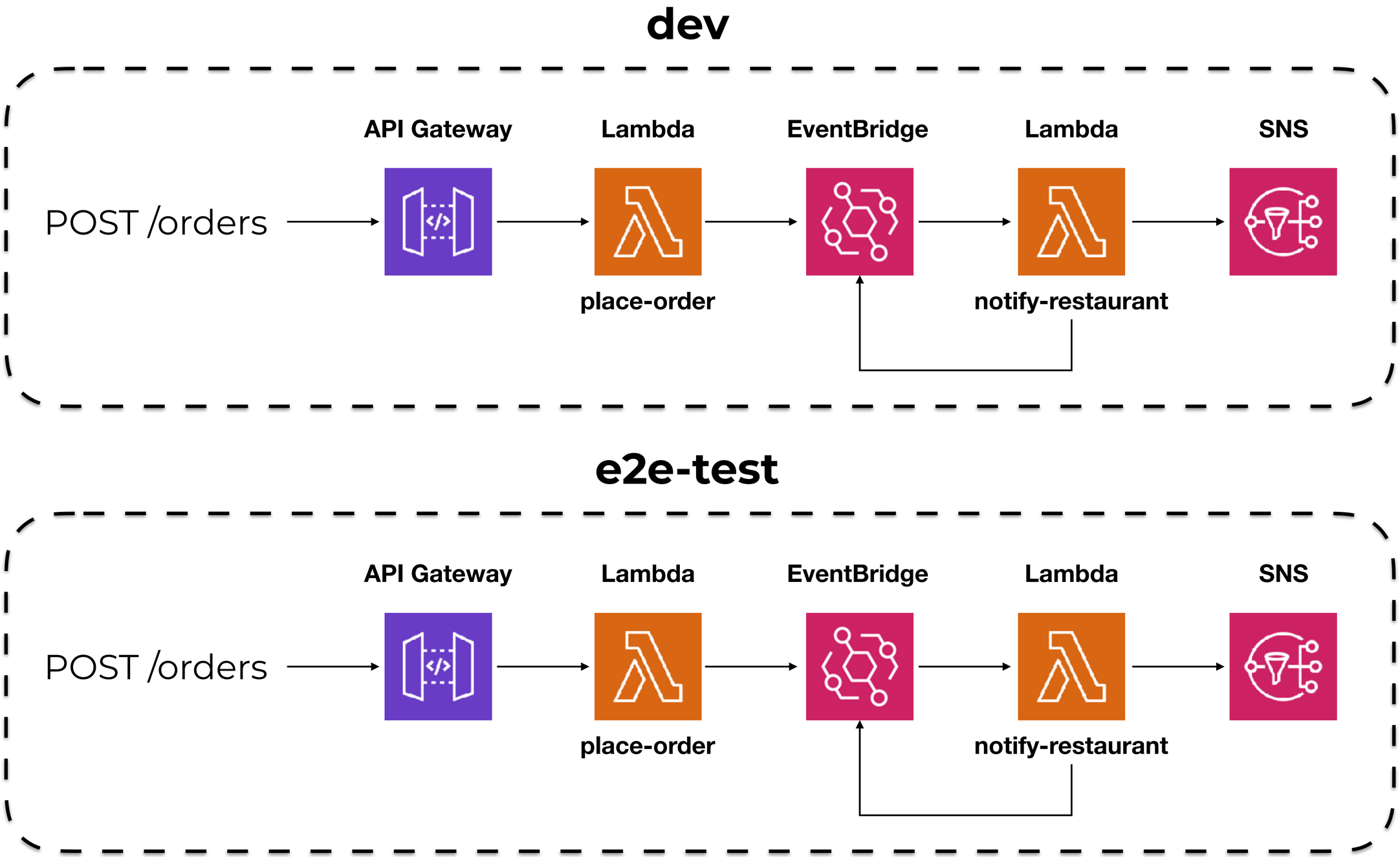
> ⚠ **Important**
>
> During a stack update, you cannot update conditions by themselves. You can update conditions only when you include changes that add, modify, or delete resources.

https://amzn.to/2Prb1KW


triggers downstream functions
API Gateway
Lambda
EventBridge
Lambda
SNS
POST /orders
place-order
notify-restaurant


isDebug: true
POST /orders
API Gateway
Lambda
place-order
isDebug: true
EventBridge
Lambda
notify-restaurant
SNS


isDebug: true
POST /orders
API Gateway
Lambda
place-order
isDebug: true
EventBridge
Lambda
notify-restaurant
SNS
BAD IDEA


dev
API Gateway
Lambda
EventBridge
Lambda
SNS
POST /orders
place-order
notify-restaurant
e2e-test
API Gateway
Lambda
EventBridge
Lambda
SNS
POST /orders
place-order
notify-restaurant

sls deploy -s e2e-test

sls remove -s e2e-test

Why you should use temporary stacks when you do serverless

AWS, CloudFormation, Programming, Serverless / September 12, 2019

One of the benefits of serverless is the pay-per-use pricing model you get from the platform. That is, if your code doesn't run, you don't pay for them!

Combined with the simplified deployment flow (compared with applications running in containers or VMs) it has enabled many teams to make use of temporary CloudFormation stacks.

In this post, let's talk about two ways you should use temporary CloudFormation stacks, and why. ***Disclaimer:*** *this shouldn't be taken as a prescription. It's a general approach that has pros and cons, which we will discuss along the way.*

Temporary stacks for feature branches

It's common for teams to have multiple AWS accounts, one for each environment. While there doesn't seem to be a consensus on how to use these environments, I tend to follow these conventions:

- dev is shared by the team, this is where the latest development changes are deployed and tested end-to-end. This environment is unstable by nature, and shouldn't be used by other teams.
- `test` is where other teams can integrate with your team's work. This environment should be fairly stable so to not slow down other teams.
- `staging` should closely resemble production, and would often contain dumps of production data. This is where you can stress test your release candidate in a production-like environment.
- and then there's `production`

https://bit.ly/3keQG9V