Phased rollouts for your iOS or Android App

Skip Ahead

Implement a phased rollout
Take action based on results

This article will help you:

• Use phased rollouts to mitigate risk of crashes
• Implement code / experiment changes to facilitate a feature rollout

Apps live and die by sprint and app store review calendars. As developers iterate, they were once bound by deadlines imposed by the app review process. Optimizely makes phased rollouts and immediate responsiveness to user engagement data possible without releasing new code to the app store.

Implementing a phased rollout for a new feature with Optimizely requires two key steps:

• Create a "Code block" within your application that will contain your feature
• Allocate traffic from your experiment options, in order to distribute this feature to a small amount of your users

Setting up a phased rollout will provide you with valuable information about how users react to your new feature, allowing you to take action on whether to release the feature to a wider audience.

Why are phased rollouts important? Let’s say you are a product manager and have spent countless hours with your team building a new design of your app. In user groups, 20 people gave positive feedback. You’re fired up to get it into the world, but not 100% of the world. You know that rolling this feature out in phases is wise because your user base is delicate, and new releases can shake things up.

Some examples of features you may want to test on a smaller audience might be:

• Adding a new like / share button. Number of shares on social media may be something you’re looking to increase, so introducing this feature as part of a phased rollout help you catch a bug that you couldn't find during testing -- if
this happens, simply use Optimizely as a kill-switch and prevent any traffic from seeing the variation with the new feature.

- A check box confirmation in signup flow. If your app now requires the user to agree to your terms and conditions before completing the signup, you can use a phased rollout to make sure your users don’t drop-off before they’ve even signed up!

---

**Implement a phased rollout**

The first step is to include your new feature inside an Optimizely Code Block.

**iOS**

You can set this up in your viewController like so:

```swift
#import <Optimizely/Optimizely.h>

// This line defines an OptimizelyCodeBlocksKey called myCheckoutBlocksKey
OptimizelyCodeBlocksKeyWithBlockNames(myNewFeatureBlocksKey, @"newFeature");

@implementation MyViewController

- (void) someFunction {
    // This line defines Code Blocks "newFeature", and a
    // default block that is executed in the case that the experiment is
    // not activated.
    [Optimizely codeBlocksWithKey:myNewFeatureBlocksKey
     blockOne:^{
        // This block contains code for the new feature
        [self setupNewFeature];
    }
     defaultBlock:^{
        // This block doesn't contain the new feature,
        // and is executed by default
    }]
}
@end
```

**Android**

For Android, you can set up a code block for an activity like so:

```java
public class CommerceActivity extends Activity {
    private static OptimizelyCodeBlock newFeature =
        Optimizely.codeBlock("Newfeature").withBranchNames("newFeature");
```
private void newFeature {
// This line defines Code Blocks "shortCheckout", "longCheckout", and a
// default block that is executed in the case that the experiment is
// not activated.
newFeature.execute(new DefaultCodeBranch() {
    @Override
    public void execute() {
        // This block doesn't contain the new feature,
        // and is executed by default
    }
}, new CodeBranch() {
    @Override
    public void execute() {
        // This block contains code for the new feature
        setupNewFeature();
    }
});
}

Save this, and then head back to the Optimizely Editor. Go to the variation tab for the variation you want to edit, select
the code blocks item at the side, and choose the "newFeature" block. You will need to make sure that the original
variation tab's code block is set to "defaultBlock."

To use a Live Variable, we recommend using a boolean Live Variable with a default value of false. This way, you can flip
the Live Variable switch to ON to enable the new feature. Below are examples of how you might implement a boolean
Live Variable.

iOS

You can set this up in your viewController like so:

```swift
#import <Optimizely/Optimizely.h>

// This line defines an OptimizelyVariableKeyForBool called newFeatureEnabled
OptimizelyVariableKeyForBool(newFeatureEnabled, NO);

@interface MyViewController :
@implementation MyViewController
- (void) someFunction {
    if ([Optimizely boolForKey:enableNewFeature]) {
        // enable the new feature
    }
}
```
For Android, you can set up a code block for an activity like so:

```java
public class CommerceActivity extends Activity {
    private static LiveVariable<Boolean> newFeatureEnabled = Optimizely.booleanForKey("newFeatureEnabled", false);

    private void newFeature {
        if (newFeatureEnabled.get()) {
            // code for new feature
        }
    }
}
```

Tip:

You may want to test your new feature on not only a small set of your visitors, but also a specific segment of your users, for example you may want to test how well new users convert on your key metric, knowing that this is the first time they will see either version. In order to achieve this you can use Custom Tags, which will allow you to target users based on variables and attributes. You can read more on how to set custom tags up for iOS experiments or for Android.

Now that the code block is ready, you can set up the experiment’s traffic allocation so that only 5% of users see the new feature. In the Editor, select Options > Traffic Allocation and change the visitors included in the experiment to 10%, leaving the original and the variation at 50%, like so:

![Traffic Allocation](image)

Before starting the experiment, make sure you’re measuring your key metric, or your Key Performance Indicator. This could be the number of registrants, or perhaps number of social shares. You can track these within Optimizely by setting up a custom event goal to fire every time a user successfully registers, or a tap goal to track when the share button has been tapped.
Take action based on results

Now that you have set up your code blocks, your traffic allocation, and your goals, you’re ready to run the experiment and start gathering data. Here are a number of outcomes, and what action you can take on them:

A successful release

Five days into the test, you have enough data to declare your new design an outstanding success with statistical significance. You can now push this new feature to all users by duplicating the experiment and allocating 100% of the traffic to the new version:

1. Pause your experiment

![Optimizely interface showing an experiment in running status](image)

2. Duplicate it to your same project

![Optimizely interface showing an experiment in paused status](image)

3. Pause all variations except the one you want your visitors to see
Inconclusive/It’s a Tie

After a number of days, you realize that your goals are under-performing or not providing the uplift you would have liked. You may want to dig into the data more, so pushing 50% of new traffic to the variation will allow you to gather more data, run it for more time and understand the cause of the initial underperformance. Your KPIs may improve over time as users familiarize themselves with the new feature.

To perform this action, go back to the Editor, select Options > Traffic Allocation, and change the total % of traffic allocated to the experiment from 10% to 50%:

A buggy release / KPI drops

You may look at your results after a reasonable amount of time and see that your "new feature" variation is performing particularly poorly. Or equally, perhaps you have reports of a show-stopping bug and upon investigation you spot a problem with the code block for that variation.

For both outcomes you can use Optimizely as a "kill-switch," by either pausing the experiment or stopping traffic flowing towards that particular variation:
Poor performance for a segment

If the new design performs poorly with one user segment but well with another, you can ship 100% traffic for a particular segment until you get to the bottom of why the release is performing poorly with the other audience.

In order to change this, you would need to update your experiment’s targeting conditions (see the above “Tip” box).