

# Ad Format Adoption Testing with Firebase

## Implementation Guide



# Contents

---

- 03 [Implementation guide](#)
- 03 [Prerequisites](#)
- 17 [Glossary](#)

## Products & features covered

---



Google  
AdMob



Firebase  
A/B Testing



Google  
Analytics



Firebase  
Remote Config



# Implementation guide

This Implementation Guide goes through implementing and testing a new Google AdMob ad format for your app, and it's using a [rewarded interstitial ad](#) as the example test case. Keep in mind, though, you can extrapolate and use these same steps to test out [other ad formats](#), too.

This guide assumes that you already use AdMob in your app and that you'd like to test whether adding *another* ad unit (with a new ad format) will have an impact on your app's revenue or other metrics. But if you don't already use AdMob in your app, that's ok! The steps in this guide can also help you understand if simply adding an ad unit to your app has an impact on your app's metrics.

**Tip:** If there's a term that you're not familiar with, check out the glossary at the end of this Implementation Guide.

## Prerequisites

- AdMob App that's linked to a Firebase App ([learn more](#))
- An iOS, Android, or Unity project with the following libraries:
  - ◆ Google Mobile Ads (AdMob) SDK
  - ◆ Firebase SDK for Google Analytics
  - ◆ Firebase Remote Config SDK

The steps in this guide will show you how and when to add these libraries. Using the latest version of each library is recommended.



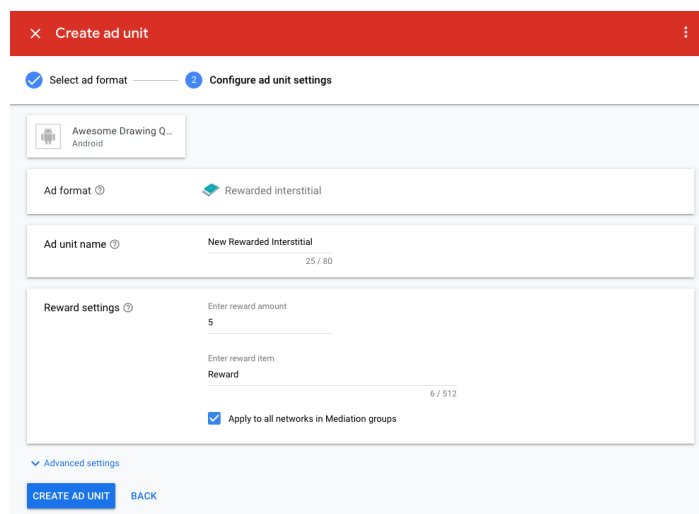
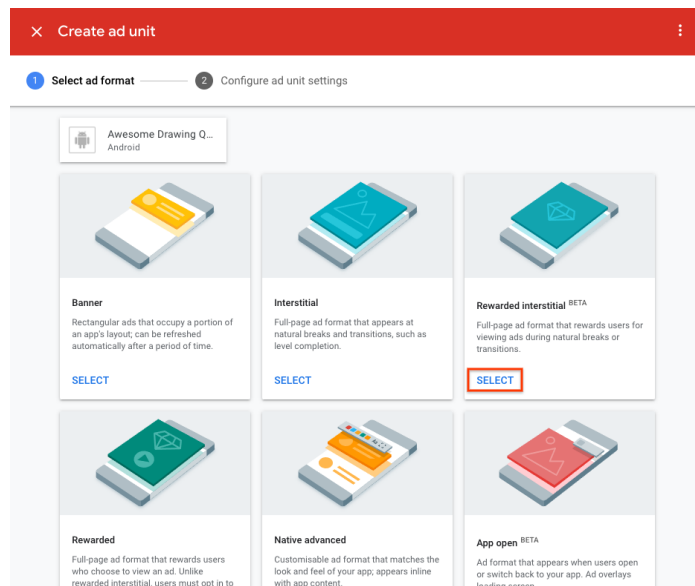
## Step 1

### ➤ Use AdMob to create a new ad unit variant for testing

This guide uses the [rewarded interstitial](#) ad format as the new format being tested for adoption. When reading this guide, though, keep in mind that you could follow similar steps to implement and test any other ad formats.

1. In your AdMob account, create the ad unit that you want to test with your users.


For this guide, create a single new ad unit with the ad format of *Rewarded interstitial*. The other ad unit settings aren't important for this particular guide, so select settings that are appropriate for your app.



2. After you create the ad unit, AdMob provides you with the ad unit's unique **ad unit ID**.

Remember where to find this ad unit ID in your AdMob account as you'll need it to implement the ad into your app.

Follow the on-screen instructions to integrate the Google Mobile Ads (AdMob) SDK and to implement the new ad unit into your app.



 **Ad unit successfully created**

Note that new ad units may take up to an hour to start showing ads. [Want to test with sample ad units while you wait?](#)

---

### Next, place the ad unit inside your app

Follow these instructions:

1. Complete the instructions in the [Google Mobile Ads SDK guide](#) using this app ID:  
 `ca-app-pub-2983478040212595-1110373942`
2. Follow the [rewarded interstitial implementation guide](#) to integrate the SDK. You'll specify ad type and placement when you integrate the code using this ad unit ID:  
 `ca-app-pub-2983478040212595/1146749882`
3. Review the [AdMob policies](#) to ensure that your implementation complies.

[EMAIL INSTRUCTIONS](#)

[DONE](#) [CREATE ANOTHER AD UNIT](#)



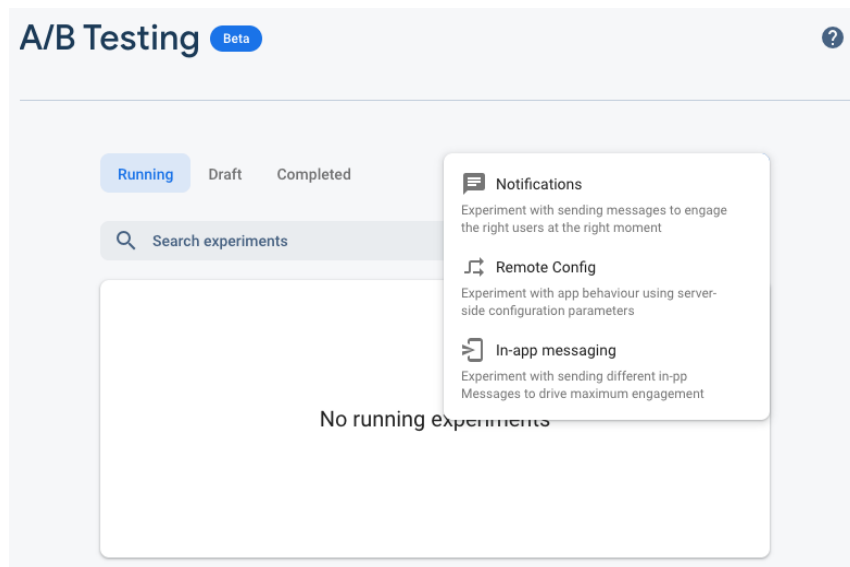
## Step 2

### ➤ Set up an A/B test in the Firebase console

Firebase A/B Testing makes it easy to test and analyze the effects of adding the rewarded interstitial ad to your app. In actuality, this testing and analyzing utilizes the following products:

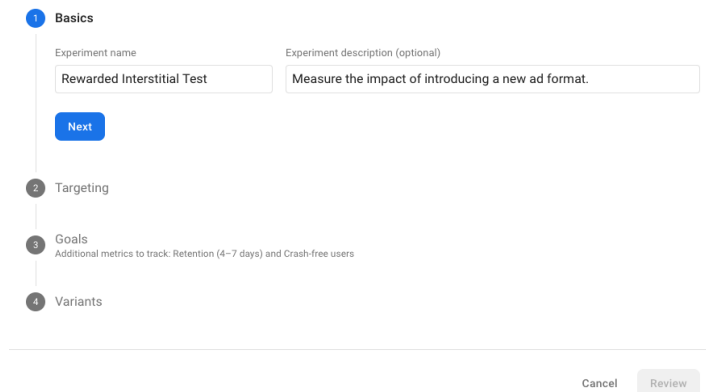
- ◆ **Firebase A/B Testing** (this step) - define goals and configurable parameters for your test
- ◆ **Firebase Remote Config** (next step) - add logic to your code to handle the configuration of the parameters
- ◆ **Google Analytics** (runs behind the scenes) - measures the impact of the configurations

To initiate a controlled test over adopting a new ad format, start by navigating to the *A/B Testing* section of the Firebase console. Click **Create experiment**, then select **Remote Config**.



## 1. Set up the basics

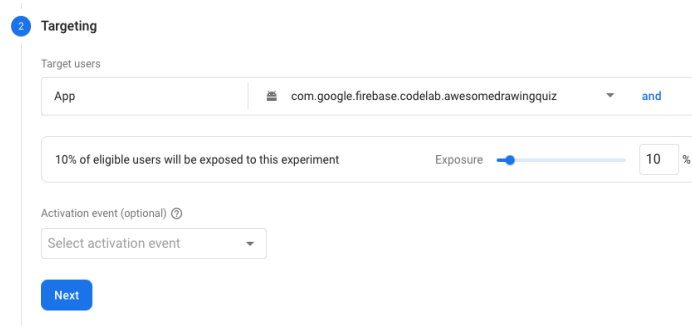
In the *Basics* section, define the experiment name and the experiment description.



The screenshot shows the 'Basics' step of the experiment setup. It features two input fields: 'Experiment name' with the value 'Rewarded Interstitial Test' and 'Experiment description (optional)' with the value 'Measure the impact of introducing a new ad format.'. A blue 'Next' button is positioned below the first field. On the left, a vertical progress indicator shows four steps: 'Basics' (active), 'Targeting', 'Goals', and 'Variants'. At the bottom right, there are 'Cancel' and 'Review' buttons.

## 2. Set up targeting

In the *Targeting* section, select the iOS or Android app (or Unity game for either of these platforms) that the experiment will target.



The screenshot shows the 'Targeting' step. Under 'Target users', there is a dropdown menu set to 'App' and a text field containing the app ID 'com.google.firebase.codelab.awesomedrawingquiz'. A blue 'and' button is to the right. Below this, an 'Exposure' slider is set to 10%, with the text '10% of eligible users will be exposed to this experiment' and 'Exposure' followed by a slider and '10 %'. An 'Activation event (optional)' dropdown is set to 'Select activation event'. A blue 'Next' button is at the bottom left.

You also need to set the percentage of users who will be exposed to the experiment. For this guide, the new ad unit will be tested with 10% of your users. Note that this doesn't mean that 10% of all your users will see the new ad format; this means that 10% of your users will be part of the experiment to see or not see the new ad format.

Leave all other settings as their defaults.

**Note:** Due to the different user behavior patterns observed from iOS and Android users, each A/B test can only target either the iOS or Android version of your app.

To run the same test for both platforms of your app (as well as for Unity games), set up an experiment for one platform, then duplicate the test settings in a second experiment. In this second experiment, select the other platform of your app in the *Targeting* section.



### 3. Set up your goals

Firestore A/B Testing tracks a primary metric to determine the winning variant, but it also allows you to add secondary metrics to understand the impacts of different configurations on other important factors for your app.

For this guide, *Estimated AdMob revenue* optimization is the primary goal, so select it from the dropdown menu. If you want A/B Testing to track additional metrics, like *Estimated total revenue* or different retention rates, select those by clicking **Add metric**.

The screenshot shows the 'Goals' configuration interface. At the top, a blue circle with the number '3' is next to the title 'Goals'. Below this, the text 'Primary metric to track (determines leader) ?' is followed by a dropdown menu containing 'Estimated AdMob revenue'. Underneath, the text 'Additional metrics to track ?' is followed by an 'Add metric +' button and four blue pill-shaped buttons: 'Estimated total revenue', 'Retention (1 day)', 'Retention (2-3 days)', and 'Retention (4-7 days)'. Each of these buttons has a small 'x' icon in its top right corner. At the bottom left of the configuration area is a blue 'Next' button.





#### 4. Set up the variants

The last step of configuring an A/B test is defining a Remote Config parameter that controls whether the new ad unit will be shown to users.

In the *Variants* section, create a new parameter named `SHOW_NEW_AD_KEY` by typing it in the *Parameter* field of the *Baseline* card.

The screenshot shows the 'Variants' section of the Firebase A/B Testing interface. It contains two variant cards. The first card, labeled 'Baseline', has a 'Parameter' field with a dropdown menu showing 'SHOW\_NEW\_AD\_KEY' and a 'Value' field with a text input containing 'false'. Below these fields is a 'Choose or create new' dropdown menu. The second card, labeled 'Variant A', has the same 'Parameter' field with 'SHOW\_NEW\_AD\_KEY' and a 'Value' field with a text input containing 'true'. Below this card is a button labeled 'Add another variant'.

For this guide, the *Baseline* variant **will not show** the new ad format to users at all, but the *Variant A* variant **will show** the new ad format to a small subset of users. This is controlled by the parameter's boolean value. These values are set here in Firebase A/B Testing, but it's Firebase Remote Config that sends these values to your app's code for handling. You'll set up Remote Config in the next step.

Complete your setup of the *Variants* section using the following settings:

- ◆ **Baseline:** *Parameter* of `SHOW_NEW_AD_KEY` set to a *Value* of `false` (which means: do **not** show new ad format)
- ◆ **Variant A:** *Parameter* of `SHOW_NEW_AD_KEY` set to a *Value* of `true` (which means: show new ad format)

**Note:** In your own future tests, if you set up various experiments and variants, we recommend giving variants meaningful names to easily track the test results later on.

Before you can start the experiment, though, you need to define how your app's code will react to the `true` or `false` parameter value received from Firebase. Proceed to the next step to implement the handling of the Remote Config parameter: `SHOW_NEW_AD_KEY`.



## Step 3

### ➤ Handle Remote Config parameter values in your app's code

#### 1. Add the required SDKs

Before using Remote Config in your application code, add both the Remote Config SDK and the Firebase SDK for Google Analytics to your project build files.

We recommend using the latest version of each SDK available, which may not be reflected in the code snippets below.

#### Android (Java) - Add the following library dependencies to your `build.gradle` file

```
implementation 'com.google.android.gms:play-services-ads:20.1.0'  
implementation 'com.google.firebase:firebase-analytics:19.0.0'  
implementation 'com.google.firebase:firebase-config:21.0.0'
```

#### iOS (Swift) - Add and install the following pods in your podfile

```
pod 'Google-Mobile-Ads-SDK'  
pod 'Firebase/Analytics'  
pod 'Firebase/RemoteConfig'
```

#### Unity - Install the Firebase Unity SDK and relevant Unity packages

[Download and install the Firebase Unity SDK](#), then add the following Unity packages to your project:

- `FirebaseAnalytics.unitypackage`
- `FirebaseRemoteConfig.unitypackage`



## 2. Configure Remote Config instance

To use the Remote Config parameter values, configure the Remote Config instance so that it is set up to fetch new values for the client app instance.

In this example, Remote Config is configured to check for new parameter values once every hour.

### Android (Java)

```
mFirebaseRemoteConfig = FirebaseRemoteConfig.getInstance();
FirebaseRemoteConfigSettings configSettings = new
FirebaseRemoteConfigSettings.Builder()
    .setMinimumFetchIntervalInSeconds(3600)
    .build();
mFirebaseRemoteConfig.setConfigSettingsAsync(configSettings);
```

### iOS (Swift)

```
remoteConfig = RemoteConfig.remoteConfig()
let settings = RemoteConfigSettings()
settings.minimumFetchInterval = 3600
remoteConfig.configSettings = settings
```

### Unity

```
var remoteConfig = FirebaseRemoteConfig.DefaultInstance;
var configSettings = new ConfigSettings {
    MinimumFetchIntervalInMilliseconds =
        (ulong)(new TimeSpan(1, 0, 0).TotalMilliseconds)
};
remoteConfig.SetConfigSettingsAsync(configSettings)
    .ContinueWithOnMainThread(task => {
        Debug.Log("Config settings confirmed");
    })
```



### 3. Fetch and activate Remote Config parameters

Fetch and activate the Remote Config parameters so that you can start using the new parameter values:

#### Android (Java)

```
mFirebaseRemoteConfig.fetchAndActivate()
    .addOnCompleteListener(this, new OnCompleteListener<Boolean>() {
        @Override
        public void onComplete(@NonNull Task<Boolean> task) {
            if (task.isSuccessful()) {
                boolean updated = task.getResult();
                Log.d(TAG, "Config params updated: " + updated);
            } else {
                Log.d(TAG, "Config params failed to update");
            }
            loadAdUnit();
        }
    });
```

#### iOS (Swift)

```
remoteConfig.fetch() { (status, error) -> Void in
    if status == .success {
        print("Config fetched!")
        self.remoteConfig.activate() { (changed, error) in
            // ...
        }
    } else {
        print("Config not fetched")
        print("Error: \(error?.localizedDescription ?? "No error available.")")
    }
    self.loadAdUnit()
}
```

#### Unity

```
remoteConfig.FetchAndActivateAsync().ContinueWithOnMainThread(task => {
    if (task.IsFaulted) {
        Debug.LogWarning("Config params failed to update");
    } else {
        Debug.Log("Config params updated: " + task.Result);
    }
    LoadAdUnit();
});
```

Your app is now ready to handle the Remote Config parameter that you created during the A/B test setup above. You'll want to make this call as early as possible in your app's loading phase because this call is asynchronous and you'll need the Remote Config value pre-fetched so that your app knows whether to show the ad.



#### 4. Use the Remote Config parameter value

Use the pre-fetched Remote Config value in the `loadAdUnit()` function to determine whether the app instance should show (parameter value of `true`) or *not* show (parameter value of `false`) the new rewarded interstitial ad unit.

##### Android (Java)

```
private void loadAdUnit() {
    boolean showNewAdFormat =
        mFirebaseRemoteConfig.getBoolean(SHOW_NEW_AD_KEY);

    if (showNewAdFormat) {
        // Load Rewarded Interstitial Ad (new implemented ad unit)
        // per AdMob instructions (the first step of this guide).
    } else {
        // Show the existing ad unit.
    }
}
```

##### iOS (Swift)

```
private func loadAdUnit() {
    let showNewAdFormat = remoteConfig["SHOW_NEW_AD_KEY"].boolValue
    if showNewAdFormat {
        // Load Rewarded Interstitial Ad (new implemented ad unit)
        // per AdMob instructions (the first step of this guide).
    } else {
        // Show the existing ad unit.
    }
}
```

##### Unity

```
void LoadAdUnit() {
    bool showNewAdFormat =
        remoteConfig.GetValue("SHOW_NEW_AD_KEY").BooleanValue;

    if (showNewAdFormat) {
        // Load Rewarded Interstitial Ad (new implemented ad unit)
        // per AdMob instructions (the first step of this guide).
    } else {
        // Show the existing ad unit.
    }
}
```

**Note:** In production code, you should also add a type and null check on retrieved Remote Config parameter values.



## 5. Add other checks for the parameter value

There are other areas in your application code where you'll need to check the value of this Remote Config parameter to instruct which ad experience will be loaded. For example, when deciding whether to reload an ad after the user has finished viewing the current one.

The fetch and activate calls should be made first to get any parameter value changes - for example if you decide to end or create a new experiment.

From there you can always check the value for the parameter using the following calls:

- ◆ Android (Java) – `mFirebaseRemoteConfig.getBoolean(SHOW_NEW_AD_KEY)`
- ◆ iOS (Swift) – `remoteConfig["showNewAdKey"].boolValue`
- ◆ Unity – `remoteConfig.GetValue("SHOW_NEW_AD_KEY").BooleanValue`

These calls will always return the same value for an app instance depending on whether it was placed in the control group or the new ad variant group, unless any changes were made in the Firebase console that were fetched and activated in the previous calls.



## Step 4

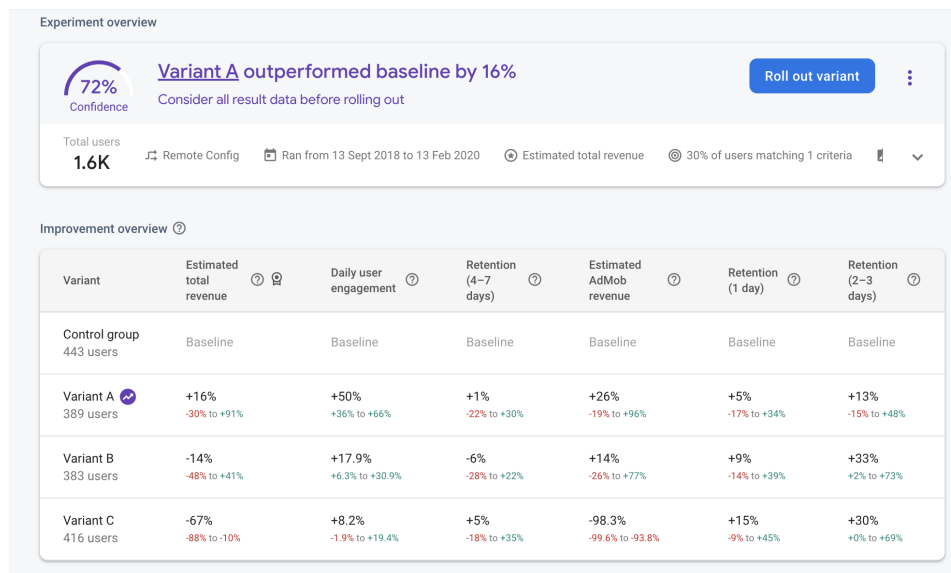
### ➤ Start the A/B test and review the test results in the Firebase console

After you add the logic to handle the Remote Config parameter value (previous step) and deploy the latest builds that include them, start the A/B test by clicking **Start Experiment** in the Firebase console.

Firestore A/B Testing will run your experiment. After it has exposed users to the different variants, the Firebase console will display an improvement suggestion. You can review how each variant performed based on the metrics that you selected during test setup.

Firestore A/B Testing makes its judgement based on the primary metric that you selected, but A/B Testing also provides you with data for all the other secondary metrics that you selected. This allows you to take into account these secondary metrics when making a final judgement about the performance of a variant.

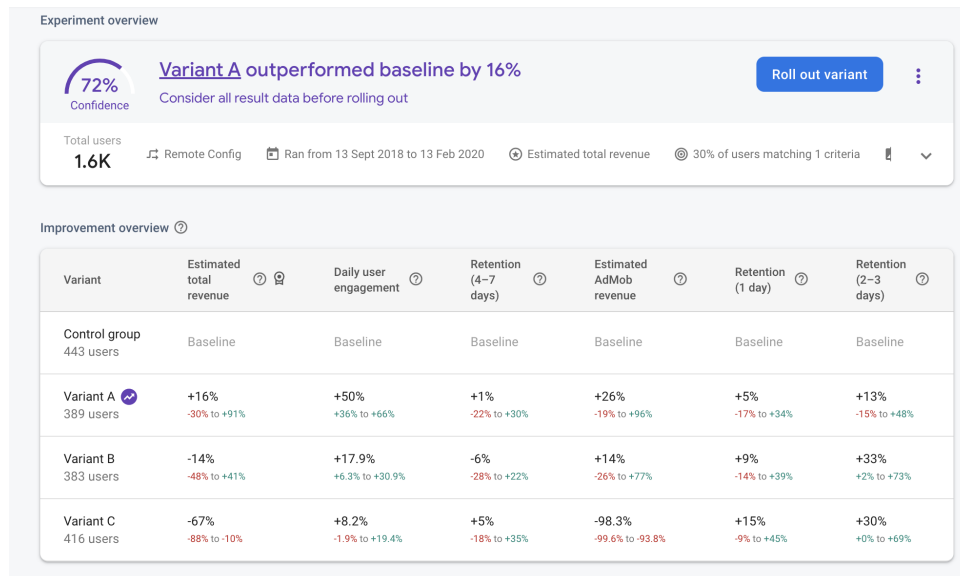
The image below shows an example of a test run with 4 variants, including the baseline (note that in this guide we kept it more simple with only two variants). A/B Testing has determined that the winning variant is *Variant A* due to the improvements in the primary metric of *Estimated total revenue*.



## Step 5

### ➤ Decide whether to roll out the new ad format

If A/B Testing determines that the variant showing the new ad format is the winner, you can start showing the ad format to all users targeted in the experiment for your app – just click the **Roll out variant** button in the A/B Testing page.



Alternatively, if a winner is determined, you can end the experiment, then set the Remote Config parameter value to the winning variant's value. You can make this be the setting for all your users or even just a subset of your users.

However, if a clear winner isn't yet determined, you can either continue running the experiment to gather more data, or end the experiment if it's already been running for a long period with inconclusive results.





# Glossary

**AdMob Revenue:** AdMob network and open bidding revenue

**IAP Revenue:** In app purchases revenue

**Total Revenue:** Total revenue

**Retention:** Retention as a key metric in A/B tests is tracked as 1-day, 2-3 days, 4-7 days, 8-14 days, or 15+ days user retention.

**Remote Config Parameter:** The configurable parameter used to control whether we show the new ad format or not. In this guide, it will be a boolean value.

**Baseline Configuration:** The as-is configuration in any particular A/B test - also known as the control. The control usually uses the default value for the Remote Config parameter, but it can be configured to use a new control value if needed.

**Variant Configurations:** The variant configurations are the alternative configurations with different Remote Config parameter values that we would like to test against the baseline configuration.



# Thank you!

